**AFRL-IF-RS-TR-2003-24**
**Final Technical Report**
**February 2003**

# POWER AWARE SIGNAL PROCESSING ENVIRONMENT (PASPE) FOR PAC/C

**Integrated Sensors, Incorporated**

**Sponsored by**
**Defense Advanced Research Projects Agency**
**DARPA Order No. J871**

**The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.**

**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-24 has been reviewed and is approved for publication.

Martin J Walter

APPROVED:

MARTIN J. WALTER
Project Engineer

James A. Collins

FOR THE DIRECTOR:

JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*

| | | |
|---|---|---|
| **1. AGENCY USE ONLY (Leave blank)** | **2. REPORT DATE**<br>FEBRUARY 2003 | **3. REPORT TYPE AND DATES COVERED**<br>Final May 00 – Nov 01 |

**4. TITLE AND SUBTITLE**
POWER AWARE SIGNAL PROCESSING ENVIRONMENT (PASPE) FOR PAC/C

**5. FUNDING NUMBERS**
C - F30602-00-C-0150
PE – 62301E
PR – PASP
TA – E0
WU – 01

**6. AUTHOR(S)**
Cosmo Castellano, Karen Solsky, John Ivory, Jim Graham

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Integrated Sensors Inc.
502 Court Street, Suite 210
Utica New York 13502

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

| | |
|---|---|
| Defense Advanced Research Projects Agency | AFRL/IFTC |
| 3701 North Fairfax Drive | 26 Electronic Parkway |
| Arlington Virginia 22203-1714 | Rome NY 13441-4514 |

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
AFRL-IF-RS-TR-2003-24

**11. SUPPLEMENTARY NOTES**

AFRL Project Engineer: Martin J. Walter/IFTC/(315) 330-4102/ Martin.Walter@rl.af.mil

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** ***(Maximum 200 Words)***

The Power Aware Signal Processing Environment (PASPE) consists of a run-time and development environment for constructing power aware applications. Power consumption is affected through management of resources, and through management of applications. Multiple application versions that trade quality of processing versus power consumption are made known to the PASPE Application Manager through the Application Registry. The Application Manager will decide whether to swap the application version based on the application request for quality of processing and real-time measurement of power supply or battery status using a user-input Mission Profile rule-set. The primary focus of this study utilized VirtexE-2000 FPGA components and compared operation of the FPGA to general-purpose processors, mainly the Intel Pentium III. Using the AFRL Flip-Wave Software Radio modem example, FPGA cores were created to measure the variation in power and energy in response to variation of processing gain. Measurement of a 32-bit floating-point complex FFT core demonstrated an 89x reduction in energy at the FPGA and 109x reduction at the FPGA board level as spectrum length was varied from 32K-point to 256-point FFT.

**14. SUBJECT TERMS**
Power, Heterogeneous Processors, Parallel Processing, Adaptive Computing Systems, FPGA, HRTExpress, WILDSTAR, FIREBIRD, COREFIRE.

**15. NUMBER OF PAGES**
94

**16. PRICE CODE**

| | | | |
|---|---|---|---|
| **17. SECURITY CLASSIFICATION OF REPORT** | **18. SECURITY CLASSIFICATION OF THIS PAGE** | **19. SECURITY CLASSIFICATION OF ABSTRACT** | **20. LIMITATION OF ABSTRACT** |
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

# TABLE OF CONTENTS

# LIST OF FIGURES

# 1 Introduction

The ISI Power Aware Signal Processing Environment Program was focused on developing a capability to allow system designers to implement orders of magnitude reduction in system power by intelligently controlling those factors that influence power consumption. ISI provided an embedded hardware configuration and software tools to support Power Aware concepts and application programming.

The project concentrated on power aware functionality in vector signal processing hardware and software. Standardized, power-conserving technologies now emerging in commercial mobile PC computing components were examined for insertion into the mobile, military signal processing system. Our research found that many of the technologies proposed were not fully implemented or supported across a wide range of platforms. These technologies are fundamental building blocks required in a new power aware system architecture, and can be subsequently extended to low power requirements for Space and Unmanned Aerial Vehicle (UAV) applications. Our target architecture was based on a commercial, off-the-shelf (COTS), power managed host computer and a modified COTS (Xilinx-Virtex based) adaptive computing board. This architecture was demonstrated on a power managed, Smart Software Radio application.

Advanced Configuration and Power Interface (ACPI), System Management Bus (SMBus), and Smart Battery are innovative standards, produced by the commercial market, for power management in mobile computing. This program evaluated these concepts to leverage and extend then to introduce a power management capability for Adaptive Computing Systems (ACS).

Power savings is an aggregate benefit of a power managed commercial host processor, teamed with a power managed, and application tailored, set of Field Programmable Gate Array (FPGA) based processors. Power management for the ACS will be accomplished through hardware modifications and control of FPGA cores, clock frequency scaling, clock gating, and sleep mode selection for ACS devices (FPGAs, memory, I/O, bus interface, etc.).

A power aware Application Programming Interface (API) was completed through specialized middleware that interfaces to the host power control functions for resource management. Middleware functions provide the user application with the ability to place unused devices or subsystems of the host computer and ACS into a reduced power, sleep or power down state. Components can be awakened or put to sleep as desired in accordance with evolving algorithmic needs. To minimize power utilization in the active portions of the ACS, the Power Aware FPGAs will be adaptively tailored to only implement, as required, the minimal needed functionality and to operate at the lowest possible clock rate required for the algorithm. This provides 100% processor utilization, resulting in the minimum required power for the job.

The Power Aware Signal Processing Environment (PASPE) will also support algorithm power adaptation. Based on remaining power estimates from an underlying Smart Battery System, the system makes automatic, real time algorithm thread swaps, based on user directed parameters. Focusing on the ACS, extraordinary potential for savings in power, as compared to currently fielded systems using general-purpose processors, is possible through FPGA implementation and ability to tailor ACS functionality at the bit level.

Voltage scaling was not addressed in this research. The Xilinx parts and board design selected did not support voltage scaling without damaging the components. The general-purpose processor hardware board design was also not capable of supporting voltage scaling. Therefore, this effort concentrated on algorithm power adaptation as described earlier.

This effort was completed and demonstrated a Power Aware Signal Processing Environment (PASPE) (see Figure 1) that addresses the mobile military signal-processing requirement as stated above. The components of this development are:

1.) A software middleware interface to provide applications with high level control of power management functions, and high level control of power management functions for an attached ACS.

2.) An application programming environment consisting of an Application Manager and Power Monitor which provide real time application thread swapping and adaptation to lower power situations. These functions implement the automatic graceful degradation of an application based on remaining battery power. Also included in this environment are Graphical User Interface (GUI) based, power profiling tools for post-mortem time base analysis of power utilization. A high-level language environment (MATLAB, H-RTExpress) will be used for rapid prototype development.

3.) A new PASPE ACS, based on a modified Annapolis Micro Systems (http://www.annapmicro.com) FIREBIRD, that will support the power management features. This new ACS utilized dynamic reconfiguration and precise tailoring of FPGA based functions and clock rates to produce the minimum required throughput at minimum power consumption.

4.) Development and demonstration of an applicable multi-mode application program that makes full use of ACS power management functionality. This program will demonstrate the use of FPGA functional tailoring, as well as decision processing based on power monitoring and mission profiles.

5.) Integration and test of the above components, using a Smart Battery simulation and host computer system. This system will allow comparison of power usage with and without use of power saving techniques. This system will become a demonstration and development platform for further HW/SW power saving advancements and for development of power aware signal processing algorithms for various applications.

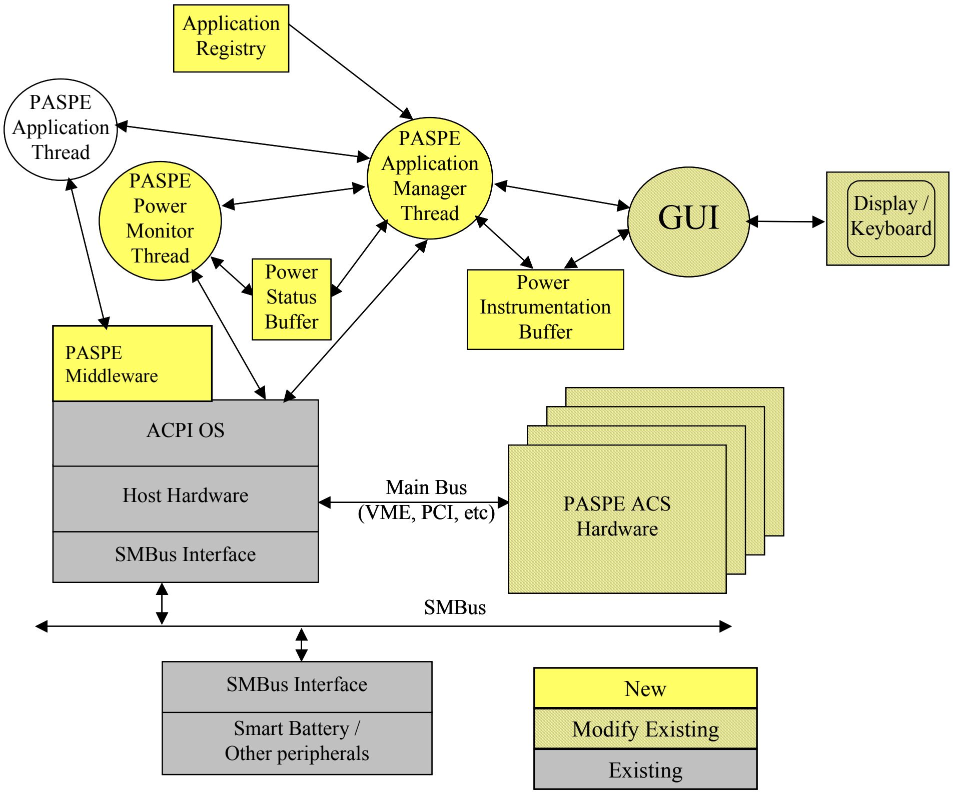


**Figure 1 PASPE Component Interfaces**

Through this effort, power will be managed through two major techniques. As shown in Figure 2, using the Application Manager and Power Monitor, the PASPE system adjusted the application's quality of service in an effort to manage power. The second dimension of power management is through management of actual resource by turning off or placing unused resources into a sleep mode, or low power state.

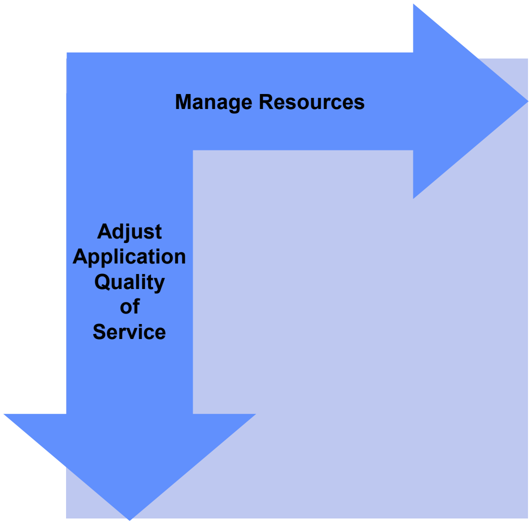


Figure 2  Two Part Approach

The stated goal of PASPE was to reduce mobile signal processor power consumption by at least 40:1 compared to currently fielded systems by creating power aware functionality in vector signal processing hardware and software.

The proposed concept examined the FPGA device to exploit its lower clock rate with the ability to spread a design across many gates, thereby accomplishing several operations in parallel.  As the clock frequency increased, the power consumed would increase, and by adjusting the bit-width used or basically adjusting the number of gate toggles per unit time, the power consumed would be adjusted.

The hypothesis is illustrated in the Figure 3 and Figure 4.   Since the General Purpose Processor (GPP) runs continuously and uses the same set of registers and devices serially, affecting the algorithm (i.e. -FFT Spectrum size), or the throughput (i.e. clock rate) is not expected to provide a great change in power consumption

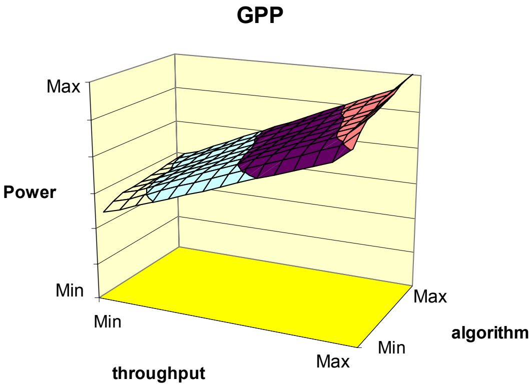


Figure 3  - Hypothesized General Purpose Processor Power Characteristics

On the other hand, using an FPGA, adjusting the toggles per time directly impacts the power consumed.

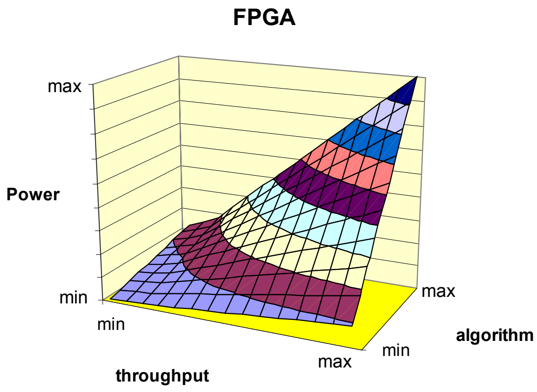


**Figure 4 - Hypothesized FPGA Power Characteristics**

Adjusting application implementations by controlling clock rate, or controlling bit-width, gating the clock or multiplexing data paths as efforts to adjust the utilization of logic gates per time unit are expected to have a direct impact on the power consumed. Also the FPGA implementation of the algorithm to be demonstrated should maximize the use of parallelism. The nature of the FPGA and the algorithm design provides a much deeper impact on power than can be expected by performing the same adjustments on the general-purpose processor.

The notion of Quality of Service as it applies to an application is the ability to adjust the application with respect to some aspect of processing quality (Signal/Noise, Gain, bit-error-rate, resolution, sample rate, etc.) The Application Manager executes the lowest power-consuming version of the system algorithm that meets the dynamic application quality of service requirement. The application is responsible for determining whether its performance is adequate and must monitor or calculate metrics to determine whether a higher quality service is required. From Figure 5, the application communicates with the application manager, and the application manager is also monitoring power availability. The application manager must decide, based on a mission profile, or rule-set, the power monitoring, and the requests from the application, which version of the application is to be run at any given time.

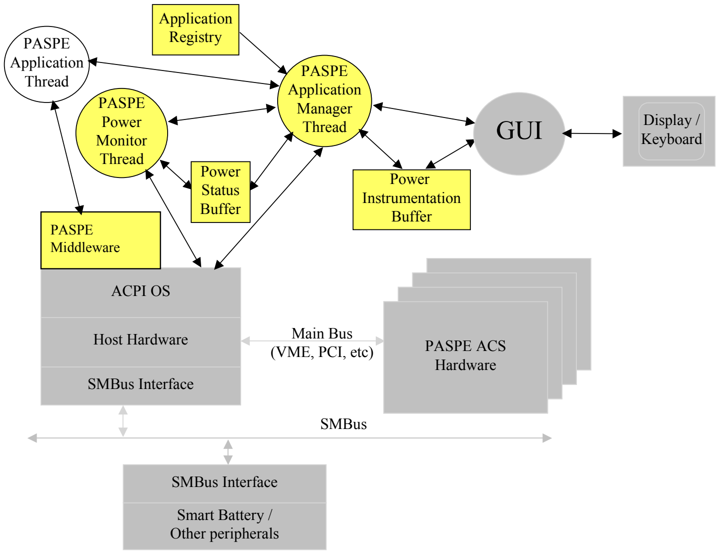


**Figure 5 - PASPE Environment and Application Quality of Service**

## 2 Architecture / Requirements

Within the Power Aware Signal Processing Environment (PASPE), power savings will be an aggregate benefit of a power managed commercial host processor, teamed with a power managed, application tailored set of Field Programmable Gate Array (FPGA) based processors. Power management for the ACS will be accomplished through utilization of hardware specific operational modes. The FPGA core images may be loaded and initialized from the host processor through the PCI interface. Although partial reconfiguration may be an important technique to consider for adjusting the hardware complexity and associated power costs, for this project the full FPGA image was loaded each time a change was requested. The ability to place the FPGA board into sleep mode, or reduced power state exists through the PCI interface, and a software controlled table is available on the FIREBIRD hardware board to specify what components are affected in each of the PCI power states.

To minimize power utilization in the active portions of the ACS, the Power Aware FPGAs may be adaptively tailored to only implement, as required, the minimal needed functionality and to operate at the lowest possible clock rate required for the algorithm. This provides 100% processor utilization, resulting in the minimum required power for the job. The PASPE environment provides this feature through algorithm power adaptation. Based on remaining power estimates from an underlying Smart Battery System, the system will make automatic, real time algorithm thread swaps, based on user directed parameters.

The PASPE environment was developed utilizing the H-RTExpress™ application development tool as a baseline. H-RTExpress™ is a software tool that assists the user in rapidly developing real-time embedded systems. Its features include support for real-time data and performance visualization, multiple parallelization paradigms, multiple heterogeneous architectures, utilization of machine specific optimized vector libraries and native compilers, utilization of FPGA core libraries, and the ability to change real-time algorithm parameters on the fly. Leveraging previous and concurrent work on H-RTExpress, this project was able to focus more directly on the power features of FPGA and programming techniques. Tool development for monitoring power or simulating battery conditions was accomplished in this program. Other software development included the Application Manager and Registry.

Figure 6 depicts the PASPE Component Interfaces and Figure 7 shows a more detailed relationship between the Application Manager Thread, the Application Registry, the Application Thread, and the GUI.

The PASPE Run-Time Environment is shown in Figure 8. The run time environment framework consists of two primary Power-Aware system tasks, the Power Monitor Thread and the Application Manager Thread. The framework also provides global mechanisms and policies for key services.

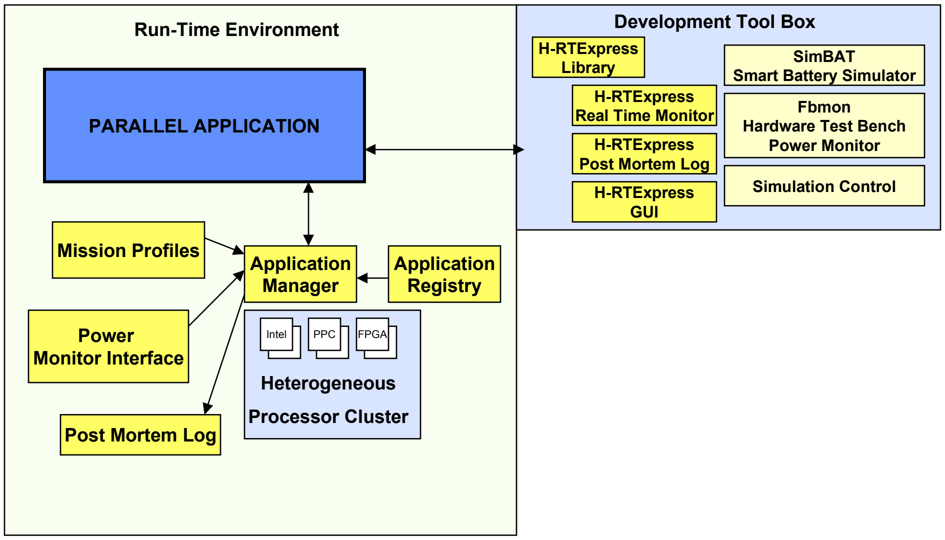
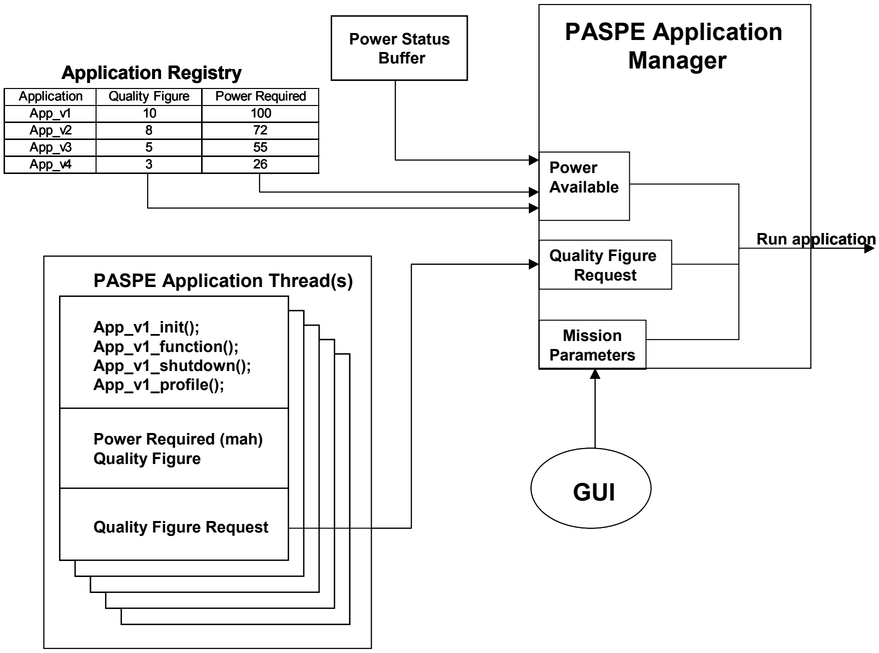


**Figure 6 PASPE Component Interfaces**



Application Registry

| Application | Quality Figure | Power Required |
|---|---|---|
| App_v1 | 10 | 100 |
| App_v2 | 8 | 72 |
| App_v3 | 5 | 55 |
| App_v4 | 3 | 26 |

**Figure 7 Detailed PASPE Component Interfaces**

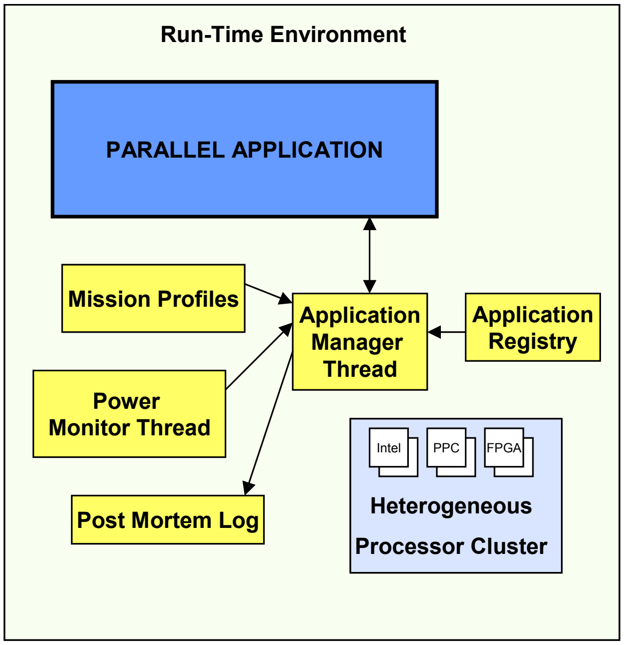


**Figure 8 PASPE Run-Time Environment**

The Application Manager provides real-time thread swapping and adaptation to lower power situations. The Application Manager has the ability to swap application versions based on requested quality, power consumption, available power, and the rules provided in the Mission Profile.

When power thresholds are detected, the Application Manager takes action to perform an orderly shutdown of the currently running application thread and transitions the application into the lowest quality version possible in order to minimize energy consumption while satisfying the mission.

The Application Manager has the ability to collect power information at programmable intervals. This information was logged into the Power Instrumentation Buffer to be used in conjunction with PASPE Viewing and Analysis tools providing post mortem analysis capability.

The Power Monitor Thread executes at a programmable interval to collect battery power information. This information is logged into the Power Status Buffer and is also shared with the Application Manager.

In addition to the periodic logging of battery power information, the Power Monitor Thread communicates with the Application Manager. It processes requests from the Application Manager (change battery query interval, battery power prediction information, etc) as well as providing unsolicited status to the Application Manager (a power threshold has been reached, etc).

An application is a collection of functions that perform a specific task such as image processing or radar signal processing. The Application Thread will contain several versions of the application allowing adaptability to the desired environment. Each version of the application has an initialization function, a processing function, and a shutdown function. The versions of the application are built as a range of “high fidelity/high power” to “low fidelity/low power” implementations.

Each version of the application has a static profile containing such information as the power required for the version and the quality figure representing the fidelity for the version. This information is stored in the Application Registry. Refer to Appendix 2 for a sample Application Registry.

## 3 Existing Commercial Standards Summary

Several existing commercial standards were suggested in the original proposal, and those standards were studied and evaluated during this research. This section summarizes that effort.

### 3.1 Smart Battery

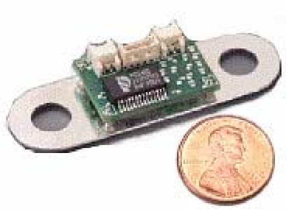


**Figure 9 - Smart Battery**

The SmartBattery concept combines a traditional battery with smart electronics. An example is shown in Figure 9. The battery itself is not smart, however, the “smarts” come from the attached electronics. Included in the “smarts” are some predictive capabilities.

The documentation reviewed was Rev 1.1 of the SmartBattery specification, dated - Dec 1998 (approx 50 pages). This standard is supported by battery manufacturers, including Duracell, Energizer, Intel, and National Semiconductor. Communication between the battery and a computer or other device is over simple 2(+) wire SMBus (I2C from Phillips). This standard allows for extreme flexibility regarding switching chemistries and battery providers.

For this research, we did not work with an actual SmartBattery, but developed a SmartBattery message simulator. The benefit over an actual battery was to not be burdened with the charge and recharge cycle of an actual battery, and to be able to construct controlled and repeatable scenarios with respect to power supply characteristics. Interfacing to an actual SmartBattery with the developed tools should be a routine replacement of the simulator with the real device.

### 3.2 SMBus

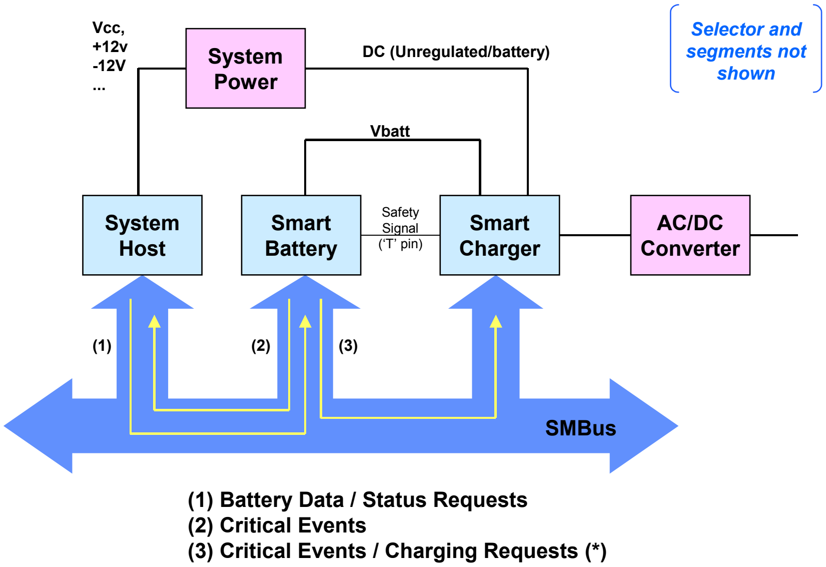


**Figure 10 - SMBus**

The System Management Bus (SMBus) is a two-wire interface for power management devices to communicate with the rest of the system. SMBus provides a control bus for system and power management related tasks. Rather than use dedicated control lines, SMBus allows messages to flow to individual devices. The advantage of SMBus is a reduction in device pin-count for dedicated control signals, and system expandability is greater since message content may grow without impact on the hardware interface. Typical messages include manufacturer information, model/part number information, save state for a suspend event, report different types of errors, accept control parameters, and return status.

The System Management Bus was originally developed by Intel, as the communication bus to accommodate Smart Batteries and other system and power management components.

SMBus has actually been in place since 1994 as part of the Onboard ACCESS bus specifications, and in 1995, Philips announced royalty free status for ACCESS bus devices. In 1996, Intel and Duracell provided Smart Battery System specifications to a group of companies forming a group known as SBS, which led to the SBS Implementers Forum in 1997. SMBus was also incorporated into the ACPI specifications in 1997 as the bus for communication with the Smart Battery System and other devices, such as temperature sensors.

The host initiates all queries to the Smart Battery System. The queries may include manufacturer, maximum capacity, battery chemistry information, granularity, status, present charge, present drain, recent drain, or temperature. Commands may be issued to define a current or power mode, or to set trigger points for alarms. The alarms are initiated by the battery system, and alarms include messages to indicate low-charge, empty, full-charge, stop charging, and over-heated.

## 3.3 ACPI

The Advanced Configuration & Power Management standard is a successor to BIOS and APM (Advanced Power Management). ACPI is not an end in itself, but a means to an end, and it is needed for operating system directed power management.

The ACPI documentation reviewed was Rev 1.0 of specification, dated - Feb 1999 (approx 325 pages).
Intel, Microsoft, Toshiba and listed as supporters, however, no updated document has been found, and support for ACPI, judging from the lack of fielded products claiming its use, appears to be minimal. ACPI is not supported in Linux, which is the operating system the PASPE project is using. ACPI also appears to be unnecessarily complicated.

Operation is logically organized into states. Global states include obvious, clearly visible states for the system as a whole. The global states are:

- G0 - Working
  Programs running
  Peripherals engaged
- G1 - Sleeping
  Minimal power use
  No program being executed
  Variable (>0) latency
  Excellent context preservation
  “Appears” to be off
- G2 - Soft off
  Large latency for power-up
  No context preservation
- G3 - Mechanical off

Device states are defined in terms of four principal criteria: Power Consumption; Device Context; Device Driver; and Restore Time. Not all devices support all four states.

- D0 - Fully on

- D1 - Better than D2,
  but worse than D0

- D2 - Better than D3,
  but worse than D1

- D3 - Electrically off
  No context preservation
  Some latency for power up

The Processing Power States exist within the global G0 (working) state. The Operating System decides when and how to kick from one processing state to another based on table info from the BIOS.
The Processing Power States include:

- C0 - Executing!

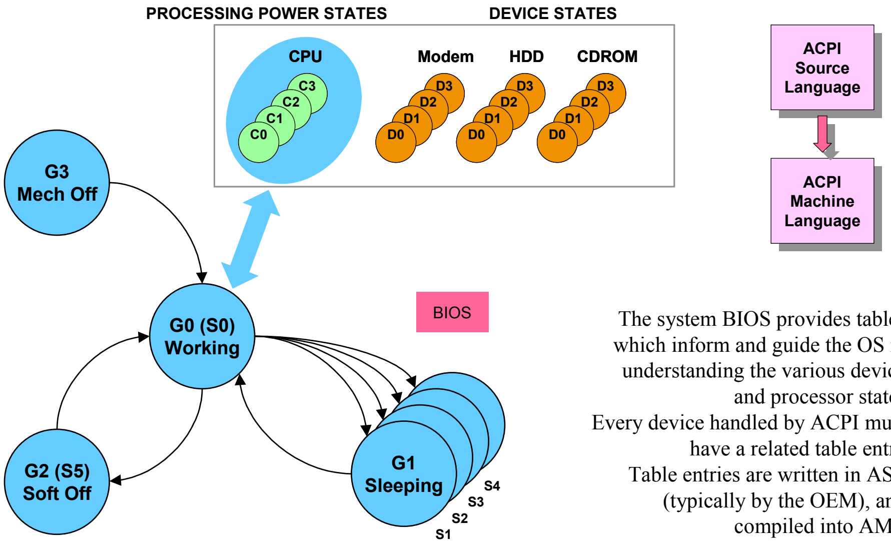
- C1 -No visible effects, but nothing is actually running. Almost immediate transition to C0. (The lights are on, but nobody's home)

- C2 - Better power usage than C1, but some latency.

- C3 - Lowest power usage. Processor caches maintain state. Some latency.

The Sleeping States exist within the global G1 (sleeping) state. No user visible work is done while in a sleeping state. The distinction between sleeping states is in what can wake the system, and how long that will take to wake the system. The Sleeping States include:

- S0 - Synonymous with G0 (Working)
- S1 - Low latency, no context loss
- S2 - Low latency, but CPU and system cache lost. Wake up from reset vector.
- S3 - All context lost, memory retained.
- S4 - Longest latency, nearly all hardware powered down
- S5 - Almost synonymous with G2 (Soft Off)



The system BIOS provides tables
which inform and guide the OS in
understanding the various device
and processor states
Every device handled by ACPI must
have a related table entry
Table entries are written in ASL
(typically by the OEM), and
compiled into AML

**Figure 11 - ACPI State Diagram**

In summary, Figure 11 depicts the overall state diagram showing the relationship between global, processing power, and device states. The system BIOS provides tables, which inform and guide the OS in understanding the various device and processor states. Every device handled by ACPI must have a related table entry. Table entries are written in ACPI Source Language (ASL), typically by the OEM, and compiled into ACPI Machine Language (AML).

Existing support for standards is presently sparse, and Linux presently provides only minimal support for ACPI. In fact, ACPI, in general, is not widely adopted. ACPI is not supported by the FIREBIRD™ device drivers.

PCI Bus Power Management is presently available in place of SMBus on the FIREBIRD FPGA board. The Power Management Register on the FIREBIRD is organized into three levels of PCI Bus Power Management (FULL, MID, LOW). Each section of the register includes a bit position to indicate which switches of the power grid are closed for each of the three states. The Power Management register is therefore a lookup table that commands the power grid based on PCI power state.

The current limitations of the commercial standards gave little impact to PASPE development.

# 4 PASPE Software Tools

To facilitate working with the FPGA hardware, manage the test scenario, and instrument voltage and current measurements from the FPGA board, graphic tools were created using the TCL/Tk language. The tools do not require the use of RTExpress™ and can communicate with each other through tcp/ip socket connections.

## 4.1 SIMBAT

### 4.1.1 What is SIMBAT?

SIMBAT is a program that simulates a smart battery.

SIMBAT starts out with a specific charge, as measured in Watt-seconds. A charge of 1000 WS would allow a drain of 10 Watts for 100 seconds, or 2 Watts for 500 seconds.

As time flows by, the program provides two strip chart displays. The top one shows what the remaining charge is. The bottom one shows what the last reported drain is. Above each strip chart is a text readout. If there is a drain on the simulated battery, the text shows an estimate of how many seconds remain before the battery runs dry.

To be more realistic, SIMBAT can simulate a natural decay. This is defined as a loss of a fixed percentage of battery capacity per second.

Warning levels and a limited variety of queries about how much energy remains in the battery are also available.

### 4.1.2 Running SIMBAT

SIMBAT is written in TCL/Tk. In order for SIMBAT to run, the machine it is executing on must have 'wish' available, and preferably be in the directory /usr/local/bin. Change directories to where the SIMBAT source files reside, and run the main shell script as follows:

```
cd ~ivory/myrt/src.hrt/simbat   (change directory to where SIMBAT source files reside)
./simbat.tcl &                  (main shell script)
```

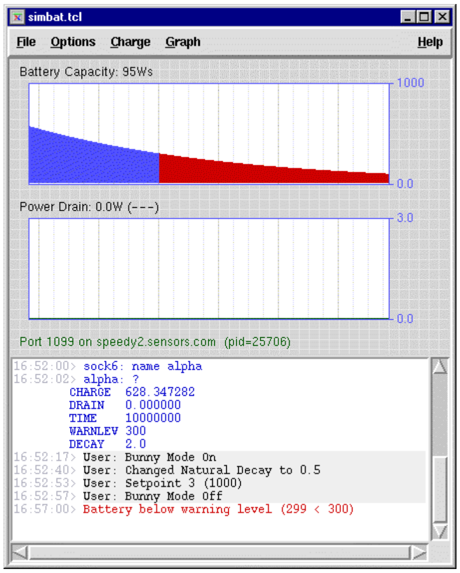
If the user wishes to run SIMBAT from a different directory, the four source files must be copied over to the new location.

Note that when SIMBAT runs, it outputs a small status file to the current directory. This means that the user must have write permission in that directory.This also opens up the possibility of usage conflicts if more than one person wants to run SIMBAT at the same time. Also, since SIMBAT puts up an X display, the DISPLAY environment must be set accordingly.

It is possible to set a few parameters by command line and environment variables. This will be covered later in this document.

### 4.1.3 The User Interface

Once SIMBAT starts, it brings up a GUI similar to the one shown in Figure 12.



Battery Capacity: 95Ws

Power Drain: 0.0W (---)

Port 1099 on speedy2.sensors.com  (pid=25706)

```
16:52:00> sock6: name alpha
16:52:02> alpha: ?
        CHARGE  628.347282
        DRAIN   0.000000
        TIME    10000000
        WARNLEV 300
        DECAY   2.0
16:52:17> User: Bunny Mode On
16:52:40> User: Changed Natural Decay to 0.5
16:52:53> User: Setpoint 3 (1000)
16:52:57> User: Bunny Mode Off
16:57:00> Battery below warning level (299 < 300)
```

This section shows a running dialog of what's been going on. Each line is time-stamped, and color provides clues about what kind of event or exchange occurred. This theme appears in a variety of the other programs, as described below.

**Figure 12 The SIMBAT User Interface**

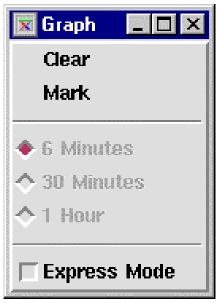
### 4.1.4 Stripcharts and Graphics

The strip charts and related text above each chart are straightforward. Vertical gray lines appear every 15 seconds, with slightly darker lines showing up to mark the minutes. If the remaining charge dips below the warning threshold, the capacity chart turns from blue to red.

When SIMBAT is in “bunny mode”, the background of the stripcharts turns a light shade of pink. When SIMBAT is in “express mode” the graphs will be nulled off and replaced with a note.

At present, the strip charts do not re-scale in either the vertical or horizontal direction (although this could be added later). If the capacity or drain exceeds the initial upper limit on the display, the graphs will max out.

Limited interaction with the stripcharts is provided. The ‘Graph’ menu option offers a tear-off menu as shown in Figure 13.



The ‘Clear’ option erases whatever is presently on both strip charts, while the ‘Mark’ options puts down a dark vertical band behind the graphs on the next tick of the clock.

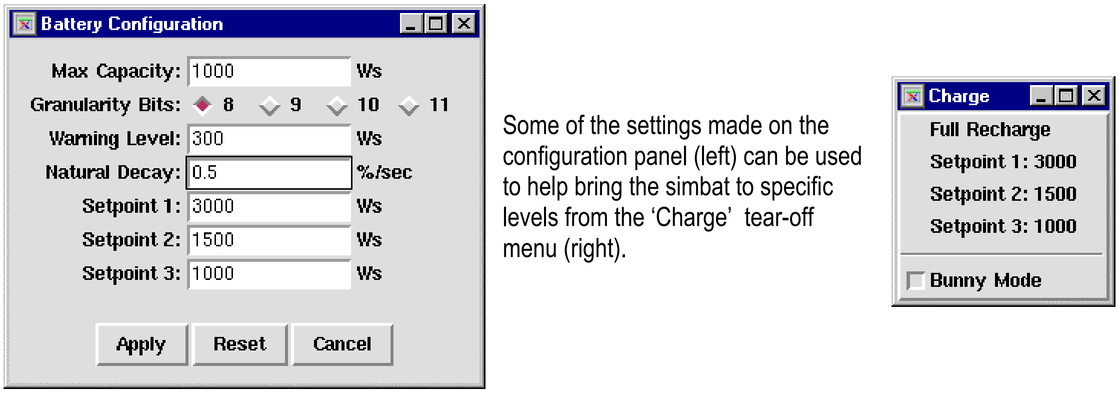
**Figure 13 Graph Menu Option Tear-Off**

The three options for time scale are grayed off. The tool is presently hard-coded to show the stripchart with a horizontal scale of 6 minutes.

SIMBAT is somewhat inefficient in how it actually generates and moves the two stripcharts. This can become a limiting factor in how fast SIMBAT runs and responds to socket communications. Putting SIMBAT into “Express Mode” turns off the management of the stripchart, allowing the tool to be more responsive.

### 4.1.5 Battery Behavior

By default, SIMBAT starts out with a maximum capacity of 4500 Watt-seconds. This, and other settings, can be changed at run-time through the “Battery Configuration” panel, found under the “Options” menu. It will bring up the display shown in Figure 14.



|  |  |  |
|---|---|---|
| Max Capacity: | 1000 | Ws |
| Granularity Bits: | ◆ 8 ◇ 9 ◇ 10 ◇ 11 |  |
| Warning Level: | 300 | Ws |
| Natural Decay: | 0.5 | %/sec |
| Setpoint 1: | 3000 | Ws |
| Setpoint 2: | 1500 | Ws |
| Setpoint 3: | 1000 | Ws |

Some of the settings made on the configuration panel (left) can be used to help bring the simbat to specific levels from the ‘Charge’ tear-off menu (right).

**Figure 14 Battery Behavior**

In addition to changing the Maximum Capacity of the battery, the user can define up to three separate charges (set points) that the battery can be quickly brought to. This makes it easy to set the charge to specific settings for testing purposes (something *not* easily done with a real battery).

The Warning Level value is the trip-point at which the graph changes from blue to red. Also, setting the Natural Decay to 0 will turn off the exponential decay of the battery.

“Bunny Mode”, as seen at the bottom of the “Charge” tear-off menu, stalls the battery capacity at whatever charge it had at the time the mode was entered. When SIMBAT is in this mode, the background of the stripcharts will be pink for the affected period. You can use the set-points to immediately set the charge to other settings, but it will not decrease as a result of time, natural decay, or drain.

It should be noted that the “Granularity Bits” option is ignored in this version of the tool (future growth).

### 4.1.6 Command Line Control and RC Files

There are two ways to control the start-up environment for SIMBAT. The first involves using a set of optional command line argument/value pairs. The possibilities are as follows:

- port ppp
  This controls the port number that SIMBAT will try to use when setting itself up for socket communications. ppp should be an integer, probably larger than 1000.
- maxcap mmm
  This sets the maximum capacity of the battery. mmm is in Watt-seconds.
- warnlev www
  This determines the warning level
- decay ddd
  Sets the ‘natural’ decay, measured as % loss per second.

It is also possible to define initial conditions for SIMBAT through use of an rc file. If a file called .simbatrc exists in the directory where SIMBAT is invoked from, it will be sourced. In order to take advantage of this option, the user should be familiar with TCL/Tk. Any combination or variations on the following lines will get the user started:

```
set simbat(def,maxcap)        4500
set simbat(def,warnlev)       900
set simbat(def,natdecay)       0.5
set simbat(def,setpoint1)     3000
set simbat(def,setpoint2)     1500
set simbat(def,setpoint3)     1000
set simbat(def,ticklen)       1000     ; # How often the stripchart changes
set simbat(def,expressmode)   0
 set simbat(drain,lo)         2.5      ; # These control the vertical
 set simbat(drain,hi)         2.75     ; # scale of the drain plot
```

### 4.1.7 Runtime Information

The bottom part of the SIMBAT GUI displays runtime information about the status of the battery, communication with clients (below), and changes made by the user. Each event is color coded and time stamped.

### 4.1.8 Socket Communications

SIMBAT communicates and interacts with other programs through sockets.

Just above the runtime information section of the GUI, there is a note indicating which machine SIMBAT is running on, what port number should be used to connect to SIMBAT (by another program), and what SIMBAT's process id is. The same information is written to a file called ".simbat.status" in the directory SIMBAT was invoked from. Programs that wish to connect to SIMBAT should open that file, and parse it to find the necessary information. SIMBAT will try to delete this status file whenever it terminates.

SIMBAT allows for multiple clients to connect at the same time, although it is expected that this number will remain fairly small.

All communications over the socket are in simple ASCII text, with each exchange being completed by an end of line (return). The client initiates all interactions. Some exchanges elicit a response from SIMBAT, some don't. There are no elaborate protocols or error checking efforts.

When multiple arguments are sent by the client, they should be separated by one or more spaces/tabs. The numeric arguments to the commands can be expressed as integers or simple floating point numbers.

The list of possible commands, and the effects they generate, are detailed below.

| **Command** | **Effect** |
|---|---|
| NAME *xxx* | A client has the option of identifying itself to SIMBAT with a name. This makes the runtime information easier to understand. |
| DRAIN *ddd* | The client is telling SIMBAT what drain (measured in Watts) it should simulate. This drain is assumed to remain in effect until changed. |
| MUTE | The client is telling SIMBAT to turn off runtime information notification whenever it changes the DRAIN. All other runtime information will still be displayed. |
| ? | The client is querying SIMBAT about its present status. The reply will be in the form of name/value pairs. Each will be on a separate line, and the pairs will be separated by a single tab. An example reply is as follows:<br>CHARGE	1104.036190<br>DRAIN	2.100000<br>TIME	95<br>WARNLEV	900<br>DECAY	3.0<br>Time is always given as an integer, estimating how many seconds remain at the present drain. If there is no drain, or SIMBAT is in Bunny Mode, the time estimate comes back as 10 million seconds. |
| HOW_LONG *ddd* | This queries the battery, asking how many seconds it thinks it can deliver the requested drain. The response is a floating point number. If the number is less than 1, then the battery already has less capacity that what's being asked for. |
| CAN_I ddd *nnn* | This is a binary question, asking SIMBAT whether or not it can deliver the requested drain for the indicated number of seconds. The reply will either be a 1 (yes) or a 0 (zero, no). |

| Command | Effect |
|---|---|
| WARNLEV *www* | This sets SIMBAT to have the specified warning level, measured in Watt-seconds. |
| CHARGE *ccc* | This sets SIMBAT to the specified charge. Note that this can be more than what the initial maximum capacity of the battery was seen to be SIMBAT started. This command is very useful for recreating test scenarios. |
| DECAY *nnn* | This sets SIMBAT to have the specified natural decay rate (measured as %/sec). |
| MARK<br>CLEAR | These act on the stripcharts, just like the options on the Graph tear-off menu, as described above. |
| BUNNY on<br>BUNNY off | Toggle in and out of Bunny Mode. |
| EXPRESS on<br>EXPRESS off | Toggle in and out of Express Mode. |
| EXPLODE | Tells SIMBAT to terminate. |
| BYE | Disconnect from SIMBAT. |

“?”, CAN_I, and HOW_LONG are the only three commands that yield a response from SIMBAT.

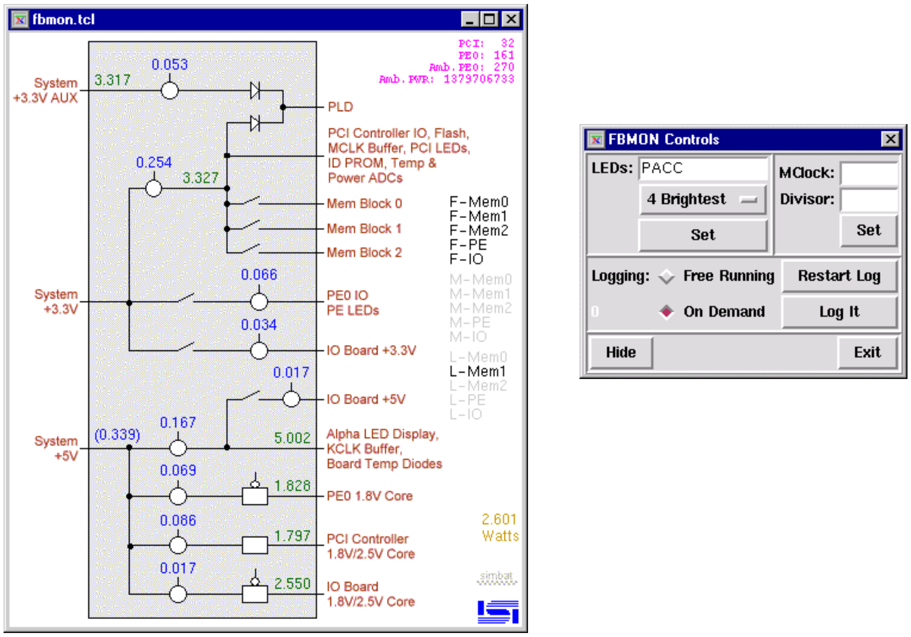
There are no disconnect or log off sequences initiated from SIMBAT. Any program connecting to SIMBAT should be prepared for SIMBAT going off-line without warning.

It is expected that there will be a background client process connected to SIMBAT which will keep it informed of the drain it is simulating (probably as a result of polling a board somewhere). A separate process can then be used to stage test conditions and poll the battery accordingly.

## 4.2 FBMON – FIREBIRD BOARD MONITOR

### 4.2.1 What is FBMON?

FBMON is a program that gives the user insight into the power circuits, power usage and temperature inside the Firebird board.



fbmon.tcl

PCI: 32
PE0: 161
Amb.PE0: 270
Amb.PWR: 1379706733

FBMON Controls

LEDs: PACC

4 Brightest

MClock:

Divisor:

Logging: ◇ Free Running

◆ On Demand

**Figure 15 FBMON**

### 4.2.2 FBMON Display

Referring to Figure 15, the gray schematic of the board, and the red text surrounding it, are just an image, redrawn from literature provided by Annapolis Microsystems. The switches shown are static representations and are not intended to show the open/closed state of the circuit.

The blue and green numbers that appear on the board are the result of direct measurement at specific points. Green numbers are voltages, and blue numbers are currents, measured in volts and amps respectively.

The displayed numbers are the result of averaging a series of measurements (presently 100 samples) at each point. Averaging helps reduce the effects of noise in the measurements.

Letting the mouse cursor linger over any of the measurements causes a small information balloon to appear, giving more detail about that measurement. Specifically, it will show the name of the measurement point, and what the standard deviation and high/low range was for the datum during the 100 sample read.

A special case for values on the board is the blue number in parentheses above the four System +5V current values. This is a calculated number representing the sum of the four current measurements below.

To get an indication of the amount of power the board is drawing, the related current/voltage pairs are multiplied and then summed. This is shown by the orange wattage readout in the lower right.

In the upper right, four different temperature measurements are shown.

Finally, down the right edge are three groups of five toggle settings. The toggle settings correspond to the control bits of the FIREBIRD Power Management Register. The bits are grouped into three categories for full, medium, and low power, which correspond to the PCI Bus Power modes. For any PCI Power Mode, the condition of the bits in that grouping determine the on or off state of each of the five switches, where a setting of logic one indicates the on mode. The five memories are controlled as three banks (Mem0, Mem1, and Mem2). The Xilinx FPGA is referred to as the processing element, or PE, and the input/output is referred to as I/O.

### 4.2.3 Running FBMON

FBMON is written in TCL/Tk. In order for FBMON to run, the user must be in the directory where the source code resides (as well as a couple of libraries and support *.gif files). Change directories to where the FBMON source files reside, and invoke FBMON as follows:

```
cd ~ivory/PA/add2tcl   (change directory to where FBMON source files reside)
fbmon.tcl &            (invoke FBMON)
```

If the user wishes to run FBMON from a different directory, the source files, libraries, and .gif files must be copied over to the new location. Also, since FBMON puts up an X display, the DISPLAY environment must be set up accordingly.

### 4.2.4 Connecting FBMON with SIMBAT

FBMON can be used as a stand-alone tool with the FIREBIRD board to observe, or to log data regarding temperature and voltage and current for all of the sensor points on the board. It is also possible to relay this information to other tools such as SIMBAT.

To run FBMON as a stand-alone tool, it is invoked without arguments.

To run FBMON in conjunction with SIMBAT, it is invoked with command line arguments used to establish a socket connection with SIMBAT.

There are two ways to do this.

```
fbmon.tcl -port <ppp> [-machine <hname>] &
```

In the first, shown above, the user specifies the port and hostname of where SIMBAT is running. This information is shown in green on the front panel of SIMBAT. The -machine argument is optional; if it is not supplied, it is assumed that the user is running FBMON on the same machine as SIMBAT.

The second method for establishing a connection relies on the status file that SIMBAT leaves behind (.simbat.status).

```
fbmon.tcl -statfile <fname> &
```

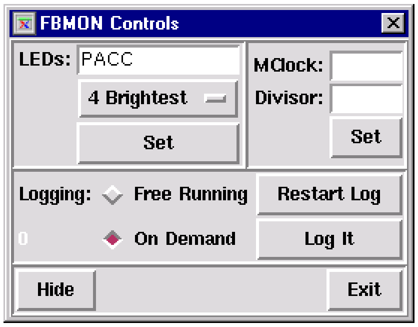
In this case, FBMON opens the indicated status file (i.e. ~ivory/myrt/src/simbat/.simbat.status), and figures out the machine and port number.

In either case, FBMON then tries to establish a socket connection. The success and status of the connectioncan be seen on the front panel of FBMON, just above the ISI logo. There will be three possibilities:

| | |
|---|---|
| simbat | fbmon was never asked to connect, and is running free. |
| simbat | fbmon is connected to simbat, and is actively communicating |
| simbat | communication is broken |

### 4.2.5 Controls and Logs

As shown earlier in this section in Figure 15, when the main FBMON display comes up, it brings along with it a small control panel. This allows the user limited interaction with the board itself, as well as the ability to invoke a logging operation for the values measured on the board.


FBMON Controls

LEDs: PACC

4 Brightest

Logging: ○ Free Running ● On Demand

0

**Figure 16 FBMON Control Dialog Box**

Referring to Figure 16, the top two sub-panels allow limited control of the board. Specifically, the left side allows the user to enter and set a 4 character code for the LED's. A simple test of the board is to play with the LED settings, and watch to see if the LED characters and brightness level changes as well as the appropriate voltage and current measurements.

The top right allows the user to set and adjust the clock for the board. It should be noted that FBMON performs no error checking/protection on the clock values entered by the user. It is possible to enter illegal values.

The middle section of this panel controls data logging for the board. The labels on the selectors are self explanatory. Basically, FBMON can be configured to make log entries on demand (via the 'Log It' button), or it can be free running. In either case, a count of how many datum have been gathered is seen in white on the left side of the sub-panel.

The log will be found in the file fbmon.log. It is an ASCII, tab-delimited file, making it extremely easy to import almost directly into a spreadsheet program. Samples of the first few rows and columns are shown below:

```
timestamp            watts        watts_pe   watts_io   mclock    V33AUX_I      ...
1007759437.692527    1.569811     0.000005   0.000000   100       0.021153
0.527                1.570229     0.000006   0.000000   100       0.021277
1.052                1.570781     0.000055   0.000000   100       0.021128
...
```

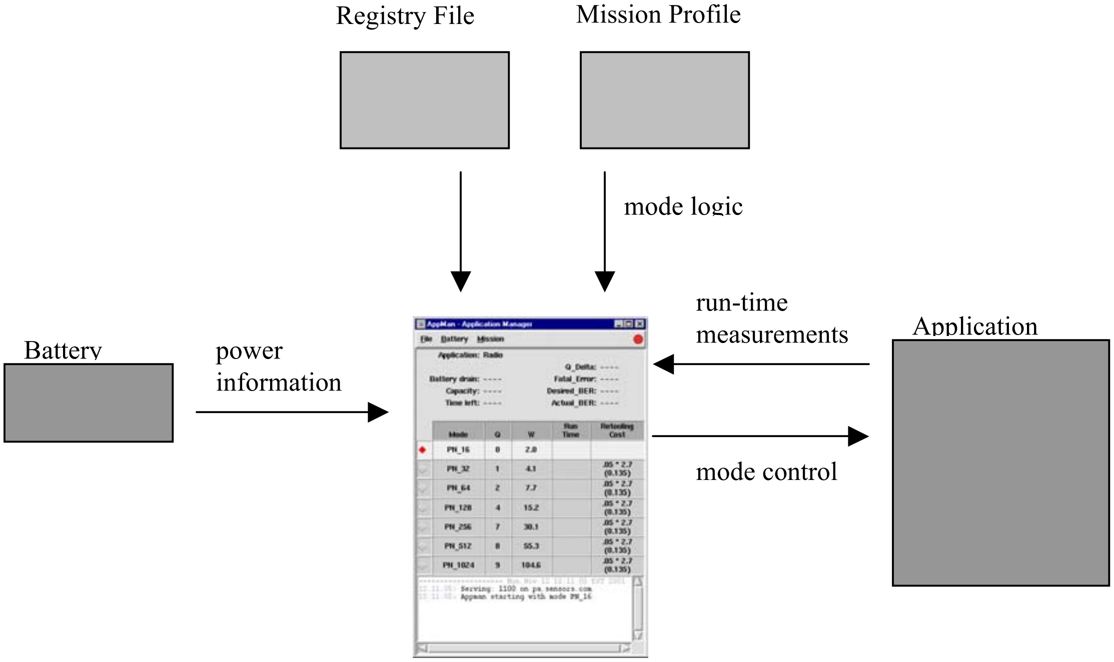
The 0'th timestamp entry shows the absolute time of when the log began. It is in standard UNIX style unix (number of seconds since the Epoch). All subsequent measurements show the elapsed time since the log began, as measured in seconds.

## 4.3 AppMan - Application Manager

### 4.3.1 What is AppMan?

The Application Manager (AppMan) acts as an overlord to a running application, making run-time mission specific decisions about which mode the application should run in. It communicates both to the application itself, and also to a (simulated) battery to get power information. An overview diagram showing the communication to and from AppMan is shown in Figure 17.

AppMan is highly configurable, and obtains all required configuration information from two separate files. The first is the application registry, which specifies what type of information will be exchanged with the application. The second file (optional) is the mission profile, which holds the decision making logic. It is possible to run AppMan in a manual mode, making selections of the desired mode directly from the column of radio-button-selectors down the left side of the display.



Application: Radio

Battery drain: ----
Capacity: ----
Time left: ----

Q_Delta: ----
Fatal_Error: ----
Desired_BER: ----
Actual_BER: ----

| | Mode | Q | W | Run Time | Reloading Cost |
|---|---|---|---|---|---|
| ◆ | PN_16 | 0 | 2.0 | | |
| ○ | PN_32 | 1 | 4.1 | | .05 * 2.7<br>(0.135) |
| ○ | PN_64 | 2 | 7.7 | | .05 * 2.7<br>(0.135) |
| ○ | PN_128 | 4 | 15.2 | | .05 * 2.7<br>(0.135) |
| ○ | PN_256 | 7 | 30.1 | | .05 * 2.7<br>(0.135) |
| ○ | PN_512 | 8 | 55.3 | | .05 * 2.7<br>(0.135) |
| ○ | PN_1024 | 9 | 104.6 | | .05 * 2.7<br>(0.135) |

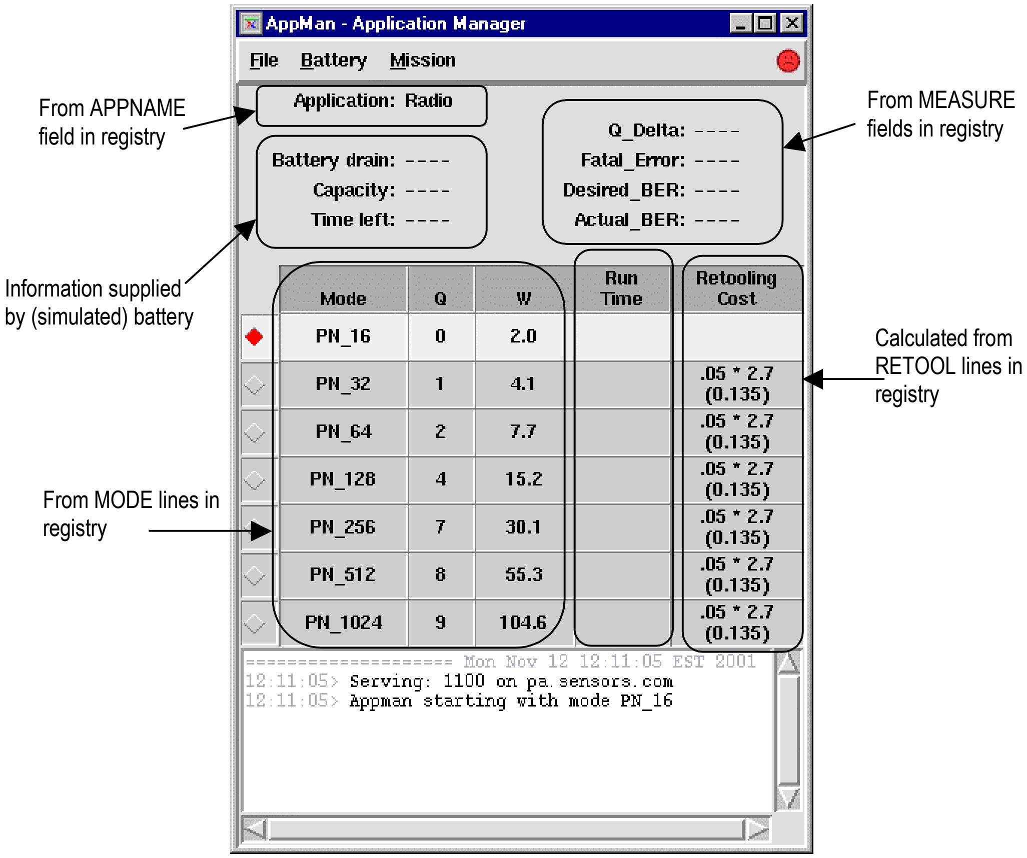
```
==================== Mon Nov 12 12:11 05 EST 2001
12.11.06: Serving: 1100 on pa.sensors.com
12.11.08: Appman starting with mode PN_16
```

**Figure 17 AppMan Overview**

Only three pieces of information are received from the battery: the present drain rate (watts), the remaining capacity of the battery (watt-seconds), and the estimated time left in the battery, assuming the present rate of drain. These datum are refreshed asynchronously once every second.

The application is free to send any subset of registered measurements to AppMan whenever it wishes. These are simply held and displayed via the GUI as they are received.

Decisions about what mode the application is to be in are transmitted from AppMan to the application only as a reply to a direct query from the application.



From APPNAME field in registry

Application: Radio

Information supplied by (simulated) battery

Battery drain: ----
Capacity: ----
Time left: ----

From MEASURE fields in registry

Q_Delta: ----
Fatal_Error: ----
Desired_BER: ----
Actual_BER: ----

From MODE lines in registry

Calculated from RETOOL lines in registry

| | Mode | Q | W | Run Time | Retooling Cost |
|---|---|---|---|---|---|
| ◆ | PN_16 | 0 | 2.0 | | |
| ◇ | PN_32 | 1 | 4.1 | | .05 * 2.7<br>(0.135) |
| ◇ | PN_64 | 2 | 7.7 | | .05 * 2.7<br>(0.135) |
| ◇ | PN_128 | 4 | 15.2 | | .05 * 2.7<br>(0.135) |
| ◇ | PN_256 | 7 | 30.1 | | .05 * 2.7<br>(0.135) |
| ◇ | PN_512 | 8 | 55.3 | | .05 * 2.7<br>(0.135) |
| ◇ | PN_1024 | 9 | 104.6 | | .05 * 2.7<br>(0.135) |

```
==================== Mon Nov 12 12:11:05 EST 2001
12:11:05> Serving: 1100 on pa.sensors.com
12:11:05> Appman starting with mode PN_16
```

**Figure 18 AppMan Main Display**

The majority of the information on the display (see Figure 18) ) is directly or indirectly available from either the registry file alone or the registry file in combination with battery data. In particular, the Run Time column is calculated on the fly. It shows an estimate of how many seconds the battery will last at the specified mode, based on the present capacity of the battery and the estimate of power necessary to run in that mode.

### 4.3.2 The Application Registry

The AppMan program gets information about the application being run from a *.reg file. A sample is shown below in Figure 19.

```
APPNAME   Radio

#
#         NAME          IDNum    Quality  Watts
#         -----         -----    -----    -----
MODE      PN_16         1        0        2.0
MODE      PN_32         2        1        4.1
MODE      PN_64         3        2        7.7
MODE      PN_128        4        4        15.2
MODE      PN_256        5        7        30.1
MODE      PN_512        6        8        55.3
MODE      PN_1024       7        9        104.6


#
#         FROM          TO       Watts    Seconds
#         -----         -----    -----    -----
RETOOL    *             *        2.7      .05


MEASURE   Q_Delta       Requested Quality Delta
MEASURE   Fatal_Error   At least one message had a fatal error
MEASURE   Desired_BER   Desired Bit Error Rate
MEASURE   Actual_BER    Actual Bit Error Rate
```

**Figure 19 - Sample Application Registry File**

In addition to comments (identified with leading '#'s), there are 4 basic types of information lines in the file.

APPNAME: This line offers a single parameter, which identifies the name of the application AppMan will be connecting with. The application must send it's own name to AppMan on initial contact, and this must match what is supplied in the registry. Since AppMan is intended to run with more than one application, this handshake helps prevent mismatches between the registry and the application.

MODE: These lines define the various modes of operation for the application. AppMan allows each mode to be identified by both a NAME and an IDNum, each of which must be unique. These in turn are uniquely associated with a measure of quality and the estimated wattage used while in that mode.

RETOOL: These lines specify the costs of switching between any combination of modes. The 'From' and 'To' fields refer to the name of the modes. Wild carding (via '*') is possible for either field.

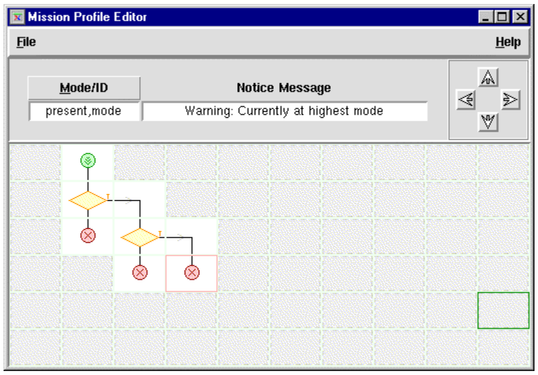
MEASURE: These lines specify the names of the various datum that will be sent from the application to AppMan during the course of the run. As they arrive, they will be posted on the top right of the GUI. The nature of the values sent from the application is completely free form; AppMan does not care if they are textual or numeric. Whatever is received by AppMan is displayed as-is. The additional information on these lines form the content of the info-bubble that will appear if the cursor lingers over the label on the GUI.

### 4.3.3 More about MODES

As mentioned above, each mode is identified by both a name and an ID number. These are largely interchangeable, and allow for flexibility in how the user and AppMan make decisions. Each mode also has a Quality rating. The lower the number, the lower the overall quality of the job being done. By being an integer value, it allows for the possibility of making more complex decisions within the mission profile. Instead of knowing only that the 'Blue' mode is better than the 'Green' mode, the quality measurement allows the user and AppMan to know that Blue is 10x better than Green.

### 4.3.4 The Mission Profile Editor

Mission profiles are stored as ASCII files, holding a pseudo language which controls the decision tree logic. Although it is possible to use a standard text editor to modify the missions, it is easier to use the editor supplied with AppMan, as seen in Figure 20.



| Mode/ID | Notice Message |
|---|---|
| present,mode | Warning: Currently at highest mode |

This section changes according to which logic cell is selected below.

**Figure 20 Mission Profile Editor**

The mechanics of using the mission profile editor are fairly straightforward and predictable.

The key points of designing a mission profile (MP) are as follows:

- Know what you want to do *before* trying to use the tool to design the logic flow.
- All logic flows begin at the top (from the single green circle) and ripple down, with each path terminating with one of the red circles.
- The current revision of the tool does not support any arithmetic functions, or complex control structures. Decisions are made based on present and prior knowledge of the state of the application.

When an application signals AppMan that it is time to make a decision, the logic flow of the mission profile is activated. The result will be for AppMan to send back to the application the new Mode/ID, and an optional text string (typically used to indicate an error condition).

It is possible to specify the resultant Mode/ID in any of a variety of ways from the editor. The button above the text field on the display brings up a cascading menu that offers the following choices displayed in Figure 21.

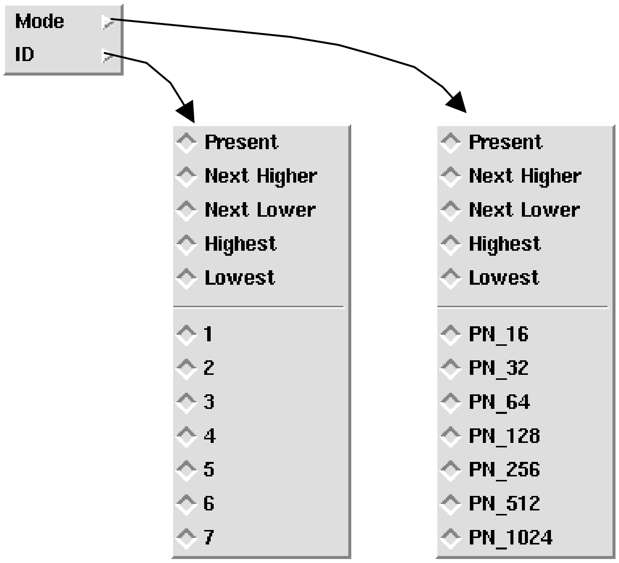
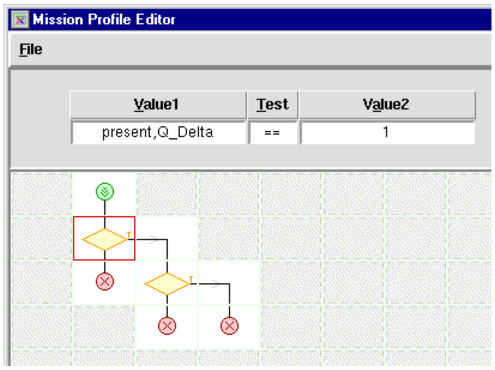


**Figure 21 Mission Profile Mode and ID Menu**

It is possible through these options to leave things unchanged (by continuing to use the 'Present' mode/id), make a relative change (next higher, lowest, etc), or an absolute change to a specific state.

### 4.3.4.1 Making decisions

In order for a mission profile to be useful, it must be possible to make decisions. When selecting a conditional logic cell from the main display, the top of the editor brings up a control panel as shown in Figure 22.



| Value1 | Test | Value2 |
|---|---|---|
| present,Q_Delta | == | 1 |

**Figure 22 Mission Profile Decision Input**

All tests are the result of a single, two argument, binary comparison. If the result of the test is true, the logic path flows to the left or right, as shown by the display. If the test is false, control flows straight down. The tests are the normal set for equality, greater than, less than, not equal, etc.

The actual arguments for the test can either be a literal value ('5', '98.6', 'HOT", etc), or a variable known to the AppMan logic. These variables fall into 5 basic categories, each of which offers a variety of options as seen in Figure 23.

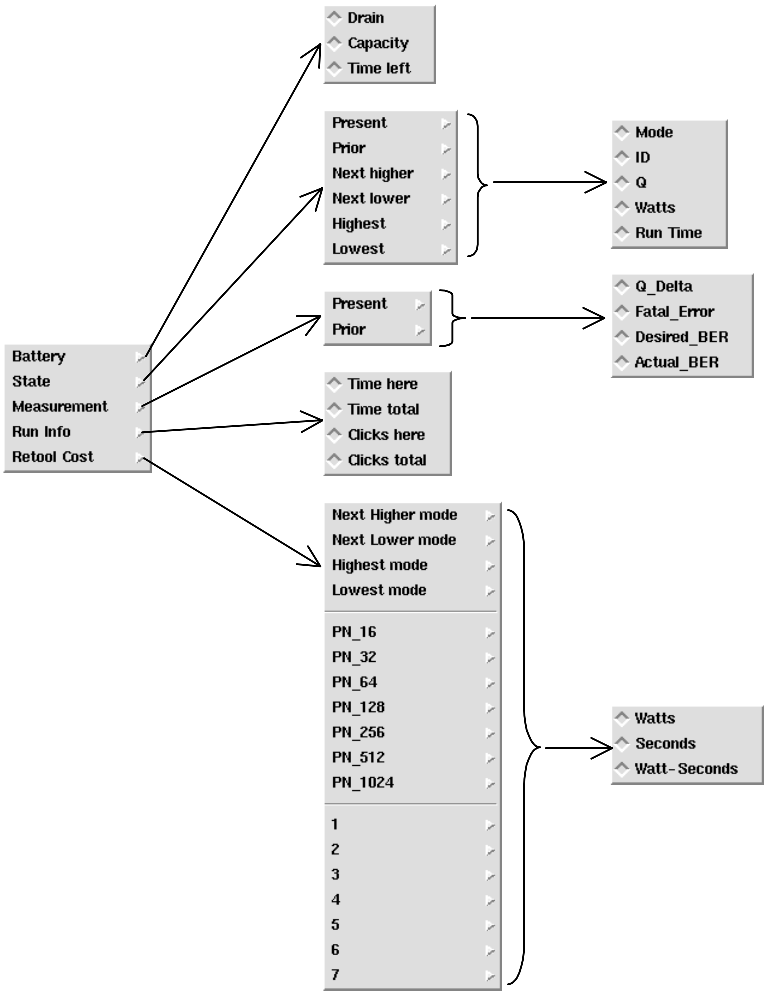


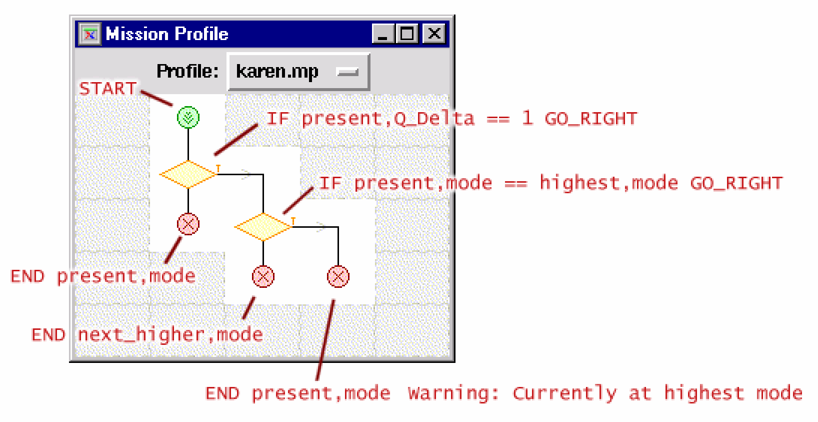
**Figure 23  AppMan Decision Variables (5 categories and options)**

### 4.3.4.2 A simple example

Working from the sample registry file in Figure 19, a simple logic tree will be created. Assume that the Q_Delta measurement is used by the application to request a switch to a higher quality mode from AppMan. Basically, the logic would read like this.

```
If the Q_delta is not equal to 1, keep the mode the same. Otherwise, go up to
the next higher mode. If we are already at the highest mode, supply a warning
message.
```

This would be diagramed as shown in Figure 24.



Mission Profile

Profile: karen.mp

START
IF present,Q_Delta == 1 GO_RIGHT
IF present,mode == highest,mode GO_RIGHT
END present,mode
END next_higher,mode
END present,mode Warning: Currently at highest mode

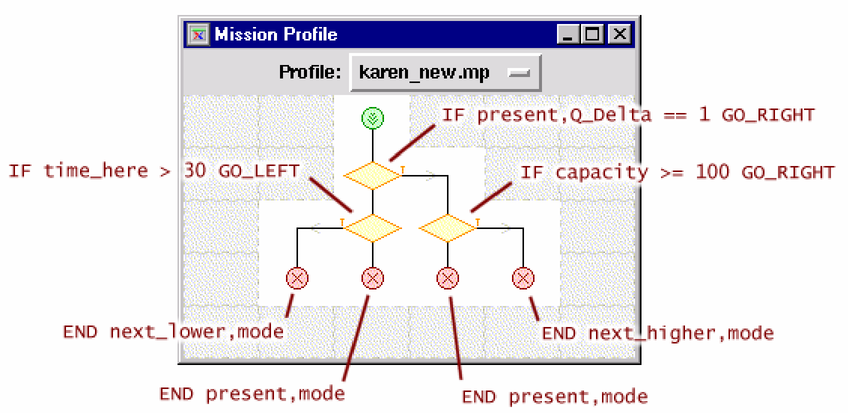
**Figure 24 Mission Profile - Example 1**

### 4.3.4.3 A second example

In this example, the remaining capacity of the battery will be taken into account and logic to transition to a lower mode will be included. The mission profile reads as follows:

```
If the application requests a higher mode (Q_Delta = 1), honor it only if there
is more than 100 Watt-Seconds of capacity in the battery; otherwise stay in the
same mode.  If the application has been in the same mode for more than 30
seconds, transition to a lower mode.
```

This is a more complex example, and is shown below in Figure 25.



Mission Profile

Profile: karen_new.mp

IF present,Q_Delta == 1 GO_RIGHT
IF time_here > 30 GO_LEFT
IF capacity >= 100 GO_RIGHT
END next_lower,mode
END next_higher,mode
END present,mode
END present,mode

**Figure 25 Mission Profile - Example 2**

Mission profiles can become quite complex. The advantage of using the mission profile editor to create profiles is that once the application registry is established, it is possible to experiment with a variety of missions without actually changing source code for either the application or AppMan.

It is also possible to change mission profiles during the running of the application itself, which is a likely event for systems deployed in the field.

# 5 HARDWARE Description

## 5.1 Annapolis Microsystems FIREBIRD PCI FPGA Board

The primary hardware used in this investigation was the FIREBIRD FPGA board (see Figure 26) produced by Annapolis Microsystems. The FIREBIRD was built with a special power-grid and current and voltage sensors(see Figure 27). The FPGA board was installed in a 64-bit / 66 MHz PCI slot of a standard Intel, Pentium III, PC running Linux, and the PC was also used in conjunction with a small, four-node Linux cluster.

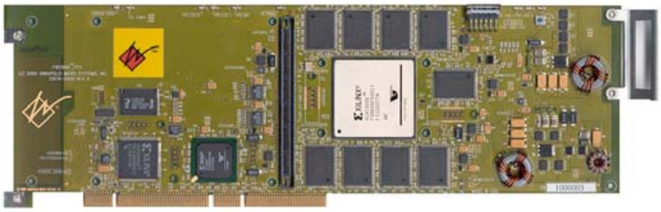


**Figure 26 Annapolis Microsystems FIREBIRD PCI board**

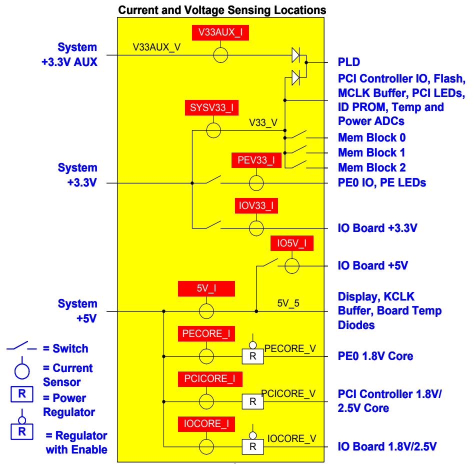
The FIREBIRD is now a commercially available product from Annapolis Microsystems, and the standard product includes the power-grid and current and voltage sensors, plus software API.

The major features of the FIREBIRD are:

- 1 Virtex™ E FPGA Processing Element
  - XCVE 1000 to XCVE2000
- 9 to 36 MBytes of Synchronous ZBT SRAM in 5 Memory Banks
- PCI Bus - Rev 2.2 Compliant
  - 5V Board - 32/64 Bit, 33 MHz, 5V or 3.3V Slot
  - 3.3V Board - 32/64 Bit, 33/66MHz, 3.3V Slot
  - Automatic 32/64 Bit PCI Bus Recognition
- Host Software: NT 4.0 and 2000, Linux, Solaris
  - API and Device Drivers
- VHDL Model of the System for Easy Development
- Accepts COTS High speed WILDSTAR™ I/O Cards
  - WILDSTAR™ Data Port (WSDPTM), FPDP, Myrinet™, 65 MHz A/D, and 1 GHz A/D

The benefits include:

- 1 to 2 Million System Gates
- Virtex™ E FPGA is larger, faster, and uses less power than Virtex™ FPGA
- 150 MHz Board, FPGA and Memory Speed
- 5.4 GBytes/Sec Memory Bandwidth
- I/O Bandwidth
  - 66 MHz PCI - Up to Theoretical Maximum of 512 MBytes/Sec with 64 Bits
  - 33 MHz PCI - 120 MBytes/Sec with 32 Bits
  - FIREBIRD™ PE to I/O Board - 3 GBytes/Sec
  - LAD Bus - 256 MBytes/Sec at 66 MHz/32 Bits
- Supports Internet Reconfiguration
- Program from Flash on Power Up
- Commercial Off the Shelf Product (COTS)



The Power Grid allows power control of the following modules:

- I/O Mezzanine
- On-Board Memory
- Processing Element
- PCI Controller

**Figure 27 FIREBIRD Power Grid and Monitoring**

Given the ability to turn power on or off to various components on the FIREBIRD board, it is essential to understand the dependencies of the board components. Figure 28 shows, for each board element, the dependency on other components of the board.

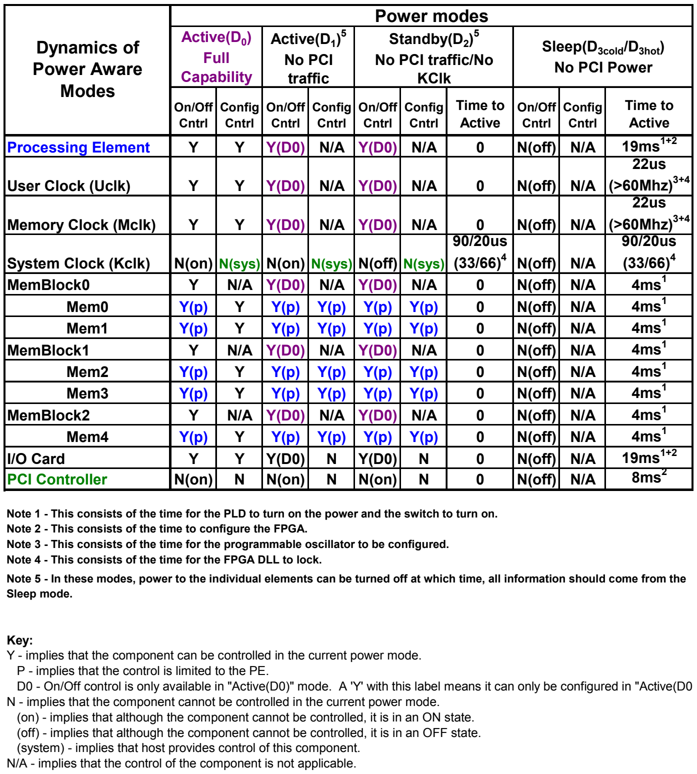
| Interdependency between Power Aware Features<br>*Depends on*<br>*Element Name* | Processing Element | User Clock (UClk) | Memory Clock (MClk) | System Clock (KClk) | MemBlock0 | Mem0 | Mem1 | MemBlock1 | Mem2 | Mem3 | MemBlock2 | Mem4 | I/O Card | PCI Controller |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Processing Element | ▨ | | ✓ | ✓ | | | | | | | | | | ✓ |
| User Clock (UClk) | | ▨ | | ✓ | | | | | | | | | | ✓ |
| Memory Clock (MClk) | | | ▨ | ✓ | | | | | | | | | | ✓ |
| System Clock (KClk) | | | | ▨ | | | | | | | | | | |
| MemBlock0 | | | | | ▨ | | | | | | | | | ✓ |
| Mem0 | ✓ | | ✓ | | ✓ | ▨ | | | | | | | | ✓ |
| Mem1 | ✓ | | ✓ | | ✓ | | ▨ | | | | | | | ✓ |
| MemBlock1 | | | | | | | | ▨ | | | | | | ✓ |
| Mem2 | ✓ | | ✓ | | | | | ✓ | ▨ | | | | | ✓ |
| Mem3 | ✓ | | ✓ | | | | | ✓ | | ▨ | | | | ✓ |
| MemBlock2 | | | | | | | | | | | ▨ | | | ✓ |
| Mem4 | ✓ | | ✓ | | | | | | | | ✓ | ▨ | | ✓ |
| I/O Card | | | ✓ | ✓ | | | | | | | ✓ | | ▨ | ✓ |
| PCI Controller | | | | ✓ | | | | | | | | | | ▨ |

✓ Indicates that the "Element Name" component has a direct dependency on the column heading component, i.e. for the yellow box, the Processing Element depends on MClk

✓ Indicates that the "Element Name" component has a indirect dependency on the column heading component

**Figure 28 - FIREBIRD Power Aware Feature Interdependency Matrix**

Power modes, as defined by Annapolis Micro Systems, are shown in the Figure 29. This chart provides the time to activate various elements from sleep mode, as well as whether a particular element may be controlled in each of the power modes D0 through D3. (Additional information on power modes is provided in Section 3.3)



| Dynamics of Power Aware Modes | Power modes | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Active(D$_0$) Full Capability | | Active(D$_1$)$^5$ No PCI traffic | | Standby(D$_2$)$^5$ No PCI traffic/No KClk | | | Sleep(D$_{3cold}$/D$_{3hot}$) No PCI Power | | |
| | On/Off Cntrl | Config Cntrl | On/Off Cntrl | Config Cntrl | On/Off Cntrl | Config Cntrl | Time to Active | On/Off Cntrl | Config Cntrl | Time to Active |
| Processing Element | Y | Y | Y(D0) | N/A | Y(D0) | N/A | 0 | N(off) | N/A | 19ms$^{1+2}$ |
| User Clock (Uclk) | Y | Y | Y(D0) | N/A | Y(D0) | N/A | 0 | N(off) | N/A | 22us (>60Mhz)$^{3+4}$ |
| Memory Clock (Mclk) | Y | Y | Y(D0) | N/A | Y(D0) | N/A | 0 | N(off) | N/A | 22us (>60Mhz)$^{3+4}$ |
| System Clock (Kclk) | N(on) | N(sys) | N(on) | N(sys) | N(off) | N(sys) | 90/20us (33/66)$^4$ | N(off) | N/A | 90/20us (33/66)$^4$ |
| MemBlock0 | Y | N/A | Y(D0) | N/A | Y(D0) | N/A | 0 | N(off) | N/A | 4ms$^1$ |
| Mem0 | Y(p) | Y | Y(p) | Y(p) | Y(p) | Y(p) | 0 | N(off) | N/A | 4ms$^1$ |
| Mem1 | Y(p) | Y | Y(p) | Y(p) | Y(p) | Y(p) | 0 | N(off) | N/A | 4ms$^1$ |
| MemBlock1 | Y | N/A | Y(D0) | N/A | Y(D0) | N/A | 0 | N(off) | N/A | 4ms$^1$ |
| Mem2 | Y(p) | Y | Y(p) | Y(p) | Y(p) | Y(p) | 0 | N(off) | N/A | 4ms$^1$ |
| Mem3 | Y(p) | Y | Y(p) | Y(p) | Y(p) | Y(p) | 0 | N(off) | N/A | 4ms$^1$ |
| MemBlock2 | Y | N/A | Y(D0) | N/A | Y(D0) | N/A | 0 | N(off) | N/A | 4ms$^1$ |
| Mem4 | Y(p) | Y | Y(p) | Y(p) | Y(p) | Y(p) | 0 | N(off) | N/A | 4ms$^1$ |
| I/O Card | Y | Y | Y(D0) | N | Y(D0) | N | 0 | N(off) | N/A | 19ms$^{1+2}$ |
| PCI Controller | N(on) | N | N(on) | N | N(on) | N | 0 | N(off) | N/A | 8ms$^2$ |

**Note 1 - This consists of the time for the PLD to turn on the power and the switch to turn on.**
**Note 2 - This consists of the time to configure the FPGA.**
**Note 3 - This consists of the time for the programmable oscillator to be configured.**
**Note 4 - This consists of the time for the FPGA DLL to lock.**
**Note 5 - In these modes, power to the individual elements can be turned off at which time, all information should come from the Sleep mode.**

**Key:**
Y - implies that the component can be controlled in the current power mode.
  P - implies that the control is limited to the PE.
  D0 - On/Off control is only available in "Active(D0)" mode. A 'Y' with this label means it can only be configured in "Active(D0
N - implies that the component cannot be controlled in the current power mode.
  (on) - implies that although the component cannot be controlled, it is in an ON state.
  (off) - implies that although the component cannot be controlled, it is in an OFF state.
  (system) - implies that host provides control of this component.
N/A - implies that the control of the component is not applicable.

**Figure 29 - FIREBIRD Power Modes**

## 5.2 ISI 4-node Linux Cluster

The FIREBIRD Board was housed in a Dell Computer chassis (see Figure 30), which was also used in conjunction with a four-node Linux Cluster containing 850MHz Pentium III processor boards. The cluster is shown in Figure 31. Each board had 512 MB of RAM, and a 100base-T Ethernet connection to the management node.

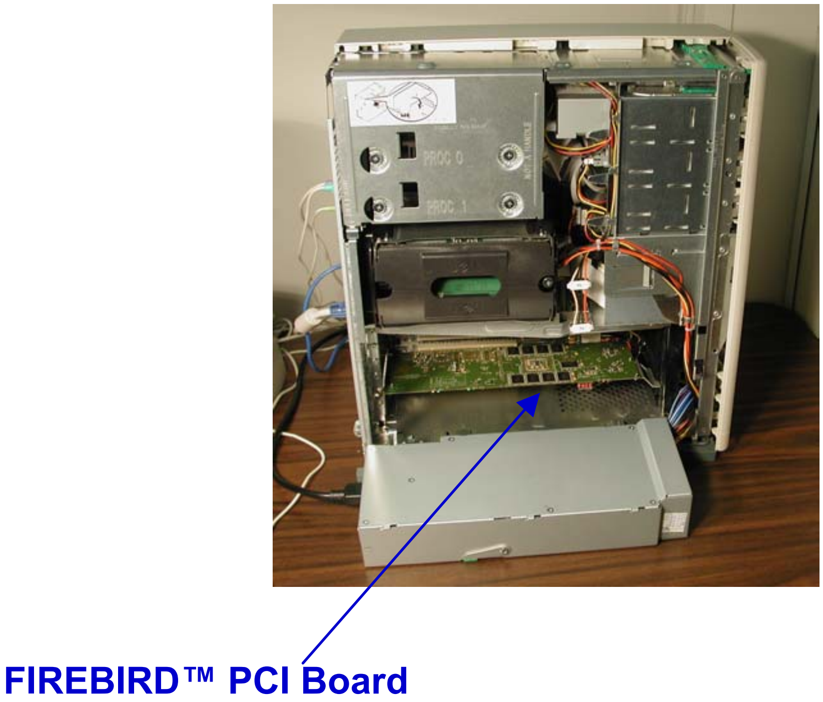


Figure 30 - FIREBIRD Board installed in PC Node

The computer node with FIREBIRD was connected using 100base-T Ethernet to the four-node Linux Cluster and its management node. Using H-RTExpress, example programs could be run solely on the PC and FIREBIRD combination chassis, or in conjunction with a parallel process in the Linux cluster.

Figure 31 - ISI Four-Node Linux Cluster

Each node of the four-node cluster featured:

Dual  650 Mhz Pentium III Processors
512 MB RAM per Node
8.4. GB SCSI Drive
Floppy Disc Drive
10/100 BT Ethernet Adapters

The nodes were contained in a roll-around rack assembly (48"), and a separate management node computer, plus monitor and keyboard was also incorporated.  Two 5 port Ethernet Switches completed the networking.  RedHat Linux, v6.2, operating system was installed with NFS file system and MPI Pro from MSTI.

Although most of the measurements took place using the FIREBIRD board and the PC it was resident in, working with the cluster allowed further splitting of a sample application.  Ideally, each node of the cluster could be enhanced with an FPGA board. Since the quantity of hardware was not available, the split problem could be studied for data communication and performance profiles using the cluster, while power related characteristics on the single FPGA board would be measured separately and combined based on the split application's profile.

## 6 Measurement Techniques

The PASPE environment assists the software developer with the ability to quickly prototype an application or code segment, and benchmark that application for performance and power characteristics.  Using the MATLAB source code language, the use of that language's "tic" and "toc" facility provides timing relative to the source commands.  Timings were also made using C-language and use of the computer clock facility.

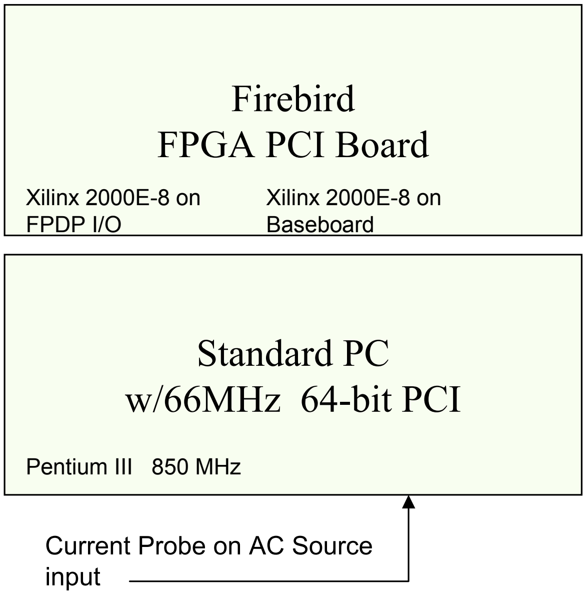

Firebird
FPGA PCI Board
Xilinx 2000E-8 on FPDP I/O
Xilinx 2000E-8 on Baseboard
Standard PC
w/66MHz 64-bit PCI
Pentium III 850 MHz
Current Probe on AC Source input

**Figure 32 - Power Measurement Approaches**

Two approaches for power measurements were taken, as shown in Figure 32.  The instrumentation points of the FIREBIRD board were used when examining the power profiles of FPGA cores.  Examination of power consumption in the general purpose processor relied on the current probe measurement technique since a similar “test point” capability did not exist in the PC. Adding current and voltage sensors to the computer motherboard were examined, however, they were abandoned since the main focus of this work was on the operation of the FPGA, and since schedule and risk did not permit altering the computer platform.

## 6.1 FPGA fbmon Power Measurements

The Annapolis Microsystems FIREBIRD board supports current and voltage measurements at several test points on the board, permitting the measurement of total power for the board, or power for the Virtex FPGA core.

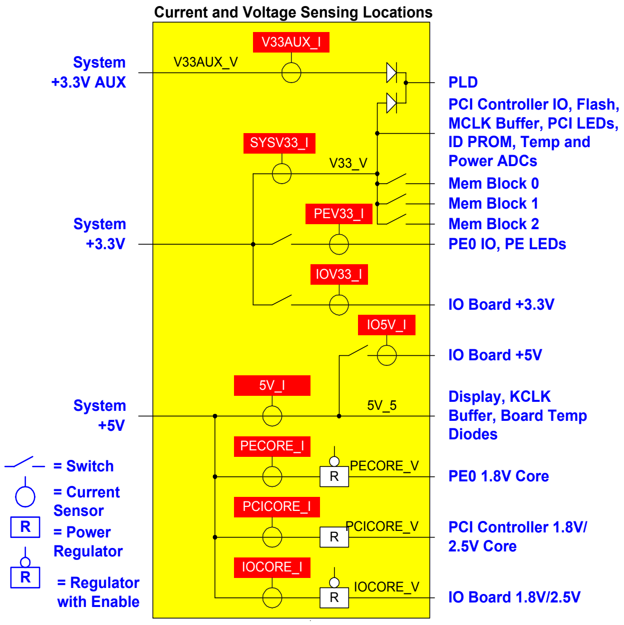
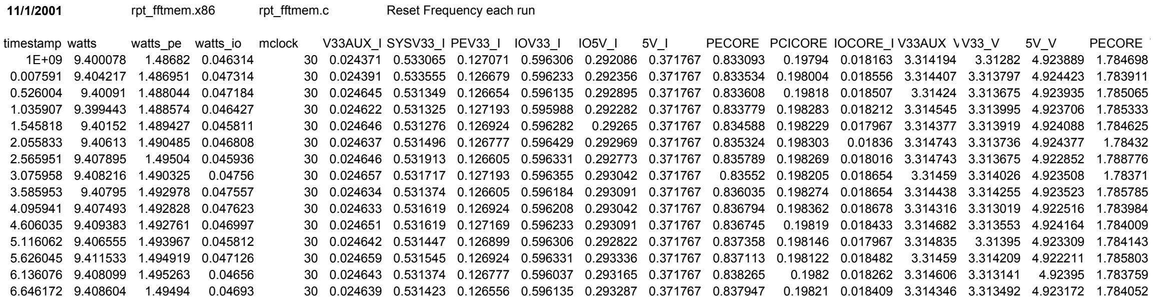


**Figure 33 - FIREBIRD Voltage and Current Sensors**

The Power Monitor software tool, fbmon, records all FIREBIRD current and voltage sensors at a programmable polling rate and logs the readings to disk. The fbmon tool runs as an independent process. The total board power is calculated by fbmon based on the summation of the currents and voltages as seen in the FIREBIRD diagram in Figure 33. Closer examination of the behavior of the FPGA is found by looking at the core power which is measured at the test points, PE0CORE_I and PE0CORE_V. Figure 34 shows a portion of the log-file created by fbmon with the time-stamped data values for a particular run.

The fbmon tool may also be controlled through tcp/ip sockets connection. For more information on fbmon, see Section 4.2.



| **11/1/2001** | | rpt_fftmem.x86 | | rpt_fftmem.c | | Reset Frequency each run | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| timestamp | watts | watts_pe | watts_io | mclock | V33AUX_I | SYSV33_I | PEV33_I | IOV33_I | IO5V_I | 5V_I | PECORE | PCICORE | IOCORE_I | V33AUX V | V33_V | 5V_V | PECORE |
| 1E+09 | 9.400078 | 1.48682 | 0.046314 | 30 | 0.024371 | 0.533065 | 0.127071 | 0.596306 | 0.292086 | 0.371767 | 0.833093 | 0.19794 | 0.018163 | 3.314194 | 3.31282 | 4.923889 | 1.784698 |
| 0.007591 | 9.404217 | 1.486951 | 0.047314 | 30 | 0.024391 | 0.533555 | 0.126679 | 0.596233 | 0.292356 | 0.371767 | 0.833534 | 0.198004 | 0.018556 | 3.314407 | 3.313797 | 4.924423 | 1.783911 |
| 0.526004 | 9.40091 | 1.488044 | 0.047184 | 30 | 0.024645 | 0.531349 | 0.126654 | 0.596135 | 0.292895 | 0.371767 | 0.833608 | 0.19818 | 0.018507 | 3.31424 | 3.313675 | 4.923935 | 1.785065 |
| 1.035907 | 9.399443 | 1.488574 | 0.046427 | 30 | 0.024622 | 0.531325 | 0.127193 | 0.595988 | 0.292282 | 0.371767 | 0.833779 | 0.198283 | 0.018212 | 3.314545 | 3.313995 | 4.923706 | 1.785333 |
| 1.545818 | 9.40152 | 1.489427 | 0.045811 | 30 | 0.024646 | 0.531276 | 0.126924 | 0.596282 | 0.29265 | 0.371767 | 0.834588 | 0.198229 | 0.017967 | 3.314377 | 3.313919 | 4.924088 | 1.784625 |
| 2.055833 | 9.40613 | 1.490485 | 0.046808 | 30 | 0.024637 | 0.531496 | 0.126777 | 0.596429 | 0.292969 | 0.371767 | 0.835324 | 0.198303 | 0.01836 | 3.314743 | 3.313736 | 4.924377 | 1.78432 |
| 2.565951 | 9.407895 | 1.49504 | 0.045936 | 30 | 0.024646 | 0.531913 | 0.126605 | 0.596331 | 0.292773 | 0.371767 | 0.835789 | 0.198269 | 0.018016 | 3.314743 | 3.313675 | 4.922852 | 1.788776 |
| 3.075958 | 9.408216 | 1.490325 | 0.04756 | 30 | 0.024657 | 0.531717 | 0.127193 | 0.596355 | 0.293042 | 0.371767 | 0.83552 | 0.198205 | 0.018654 | 3.31459 | 3.314026 | 4.923508 | 1.78371 |
| 3.585953 | 9.40795 | 1.492978 | 0.047557 | 30 | 0.024634 | 0.531374 | 0.126605 | 0.596184 | 0.293091 | 0.371767 | 0.836035 | 0.198274 | 0.018654 | 3.314438 | 3.314255 | 4.923523 | 1.785785 |
| 4.095941 | 9.407493 | 1.492828 | 0.047623 | 30 | 0.024633 | 0.531619 | 0.126924 | 0.596208 | 0.293042 | 0.371767 | 0.836794 | 0.198362 | 0.018678 | 3.314316 | 3.313019 | 4.922516 | 1.783984 |
| 4.606035 | 9.409383 | 1.492761 | 0.046997 | 30 | 0.024651 | 0.531619 | 0.127169 | 0.596233 | 0.293091 | 0.371767 | 0.836745 | 0.19819 | 0.018433 | 3.314682 | 3.313553 | 4.924164 | 1.784009 |
| 5.116062 | 9.406555 | 1.493967 | 0.045812 | 30 | 0.024642 | 0.531447 | 0.126899 | 0.596306 | 0.292822 | 0.371767 | 0.837358 | 0.198146 | 0.017967 | 3.314835 | 3.31395 | 4.923309 | 1.784143 |
| 5.626045 | 9.411533 | 1.494919 | 0.047126 | 30 | 0.024659 | 0.531545 | 0.126924 | 0.596331 | 0.293336 | 0.371767 | 0.837113 | 0.198122 | 0.018482 | 3.31459 | 3.314209 | 4.922211 | 1.785803 |
| 6.136076 | 9.408099 | 1.495263 | 0.04656 | 30 | 0.024643 | 0.531374 | 0.126777 | 0.596037 | 0.293165 | 0.371767 | 0.838265 | 0.1982 | 0.018262 | 3.314606 | 3.313141 | 4.92395 | 1.783759 |
| 6.646172 | 9.408604 | 1.49494 | 0.04693 | 30 | 0.024639 | 0.531423 | 0.126556 | 0.596135 | 0.293287 | 0.371767 | 0.837947 | 0.19821 | 0.018409 | 3.314346 | 3.313492 | 4.923172 | 1.784052 |

**Figure 34 - Sample "fbmon" log file**

### 6.2 General Purpose Processor (GPP) – Power Measurements

The standard PC chassis that the FIREBIRD was installed in did not have any built in instrumentation for gathering current or voltage test points in the same way the FIREBIRD board did. An investigation was taken to find a path to add instrumentation points that gave solutions that were either too costly or too risky. Rather than modify the computer motherboard, or interfere with the processor operation, power measurement of the general-purpose processor was accomplished by looking at the input power to the computer system.

Average power of the application under test was measured as a "delta" measurement, comparing the application power to the idle power. First, executing the code in an infinite loop, the average power measurement was recorded while running. Next, that power reading was subtracted from the average power measurement made with processor at an idle state. The resulting difference is the power attributed to the test code. The quiescent average power measurement was made on the computer was with a "safe" core loaded in the FIREBIRD FPGAs and without a software application executing on the computer.

The ability to directly compare power results between the FPGA and GPP is lost with this approach, however, relative gains in FPGA implementations with comparison to "baseline" measurements on the PC can be made. Using the "delta" power measurement technique on the GPP allowed comparison of the power consumption trend as variations of an algorithm implementation are executed. The power consumption behavior was also compared to that of the FPGA.

## 7 H-RTExpress Tool Set

This program leveraged the DARPA/TTO Adaptive Computing Systems effort (BAA 97-06). Hetrogeneous-RTExpress was contructed under Contract No. F30602-97-C-0259, and it provides support for the USC/ISI SLAAC FPGA board and Annapolis Wildstar FPGA, both for VME systems. Support for the Annapolis FIREBIRD board was easily added, and the PASPE project benefited by the use of the H-RTExpress tool for rapidly creating parallel applications and test programs with a Linux cluster and FPGA.

FPGA Core development with our partners on PASPE, Annapolis Microsystems, was accomplished using their newly created tool, COREFIRE. This tool provides a dataflow graphic programming interface with built in intellectual property in the form of parameterized cores. The speed up in FPGA development time was dramatic, and the constructed functions were easily inserted into the H-RTExpress FPGA library for use as called functions from the MATLAB source code.

The major benefits include: Integrated Graphical User Interface (GUI) for application development on heterogeneous processors (PowerPC, FPGA); GUI to provide both Hardware and Software Visualization; Reduction in time, effort, and skill level to program adaptive computing hardware; the ability to Map MATLAB® code onto heterogeneous processing nodes; and the ability to easily launch applications across heterogeneous processor architectures.

### 7.1 H-RTExpress

A heterogeneous software development environment is needed to support the programming of systems composed of both general-purpose processors such as the PowerPC and specialty processors such as WILDSTAR FPGA boards. Currently, to program a heterogeneous hardware architecture consisting of FPGA's requires the expertise of a hardware designer. Current available software programming environments do not provide a true high-level programming language to use making the developer rely on hardware design level tools and debugging at the circuit level. The software development costs and maintenance costs for FPGA boards are higher than for programming general-purpose hardware architecture.

The approach used provides a MATLAB® high-level language interface for defining an application that will be targeted to a heterogeneous hardware architecture containing FPGA's. The tools developed provide the ability at the MATLAB level to specify the inherent parallelism that exists in the application and to provide the capability to visualize the targeted hardware architecture and graphically map software components to hardware processing nodes. This approach provides for automatic generation of parallel C code for application segments targeted to the general purpose nodes and calls to preprogrammed FPGA library cores for application segments targeted to the FPGA nodes.

The H-RTExpress™ heterogeneous software architecture environment was built upon the Integrated Sensors, Inc. (ISI) commercial RTExpress™ product. The realized heterogeneous software development (H-RTExpress™) environment is illustrated in Figure 35.

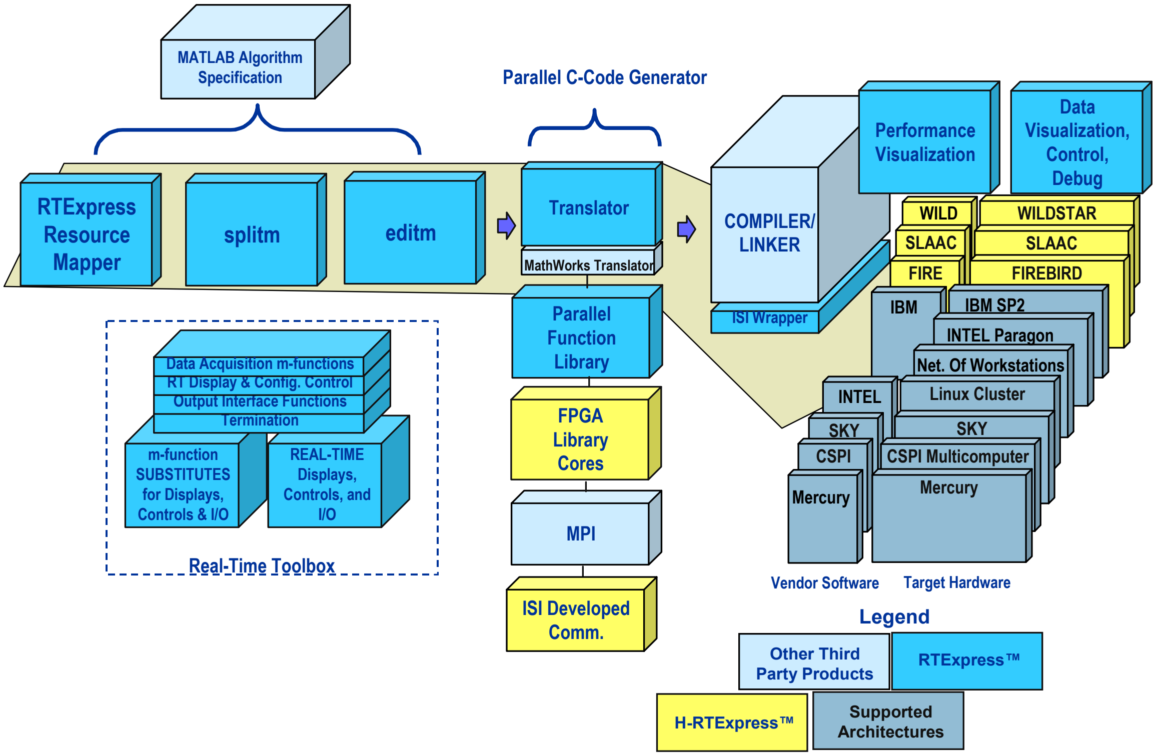
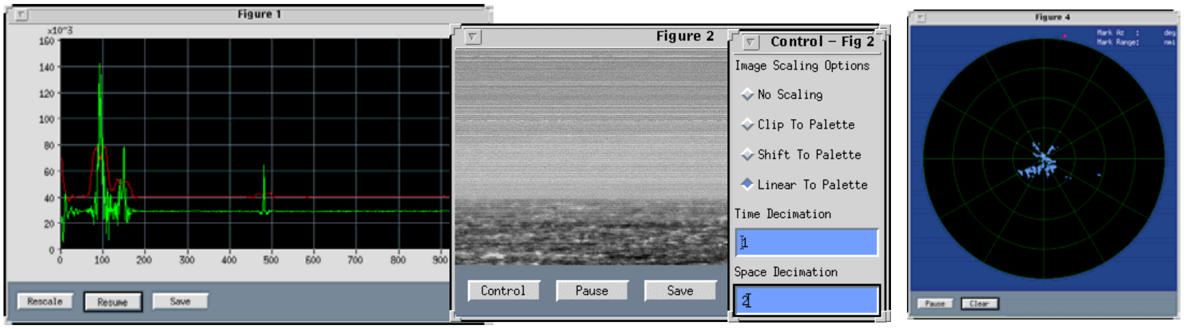

MATLAB Algorithm Specification
Parallel C-Code Generator
RTExpress Resource Mapper
splitm
editm
Translator
MathWorks Translator
COMPILER/ LINKER
ISI Wrapper
Performance Visualization
Data Visualization, Control, Debug
WILD
WILDSTAR
SLAAC
SLAAC
FIRE
FIREBIRD
IBM
IBM SP2
INTEL Paragon
Net. Of Workstations
INTEL
Linux Cluster
SKY
SKY
CSPI
CSPI Multicomputer
Mercury
Mercury
Vendor Software
Target Hardware
Parallel Function Library
FPGA Library Cores
MPI
ISI Developed Comm.
Data Acquisition m-functions
RT Display & Config. Control
Output Interface Functions
Termination
m-function SUBSTITUTES for Displays, Controls & I/O
REAL-TIME Displays, Controls, and I/O
Real-Time Toolbox
Legend
Other Third Party Products
RTExpress™
H-RTExpress™
Supported Architectures

**Figure 35 - H-RTExpress Environment**

All of the existing features of RTExpress™ are maintained in H-RTExpress™, and support for both the Annapolis Micro Systems WILDSTAR™ and USC/ISI SLAAC2 Adaptive Computing Systems boards exist. Through the PAC/C effort support for the Annapolis Microsystems FIREBIRD™ PCI board has been added, and by mid-2002, support for a Mercury RACE++ FPGA Daughter Card will be added (BAA 97-06 ECP).

The H-RTExpress™ Environment utilizes three main user tools: splitm (MATLAB editor and splitting); editm (MATLAB editor); and Mapit (resource mapper). In addition to other third party products, such as the MathWorks MATLAB Translator, MSTI's MPI Pro, and standard compiler and linker, H-RTExpress™ includes the ISI Real-Time Toolbox functions and parallel function library that brings MATLAB functions to a parallel implementation. Where available, vendor supplied libraries are used, as is the case with the Mercury Computer Scientific Application Library (SAL).

Real time data visualization and user defined buttons and input and output data boxes are available in the RTExpress™ Real-Time Toolbox graphic user interface features as shown in Figure 36.



Emulates MATLAB® plot with enhancements for real-time operation

**Data Pause/Save to disk for User Analysis**
**Variable scaling**
**Time Decimation to control I/O Loading**

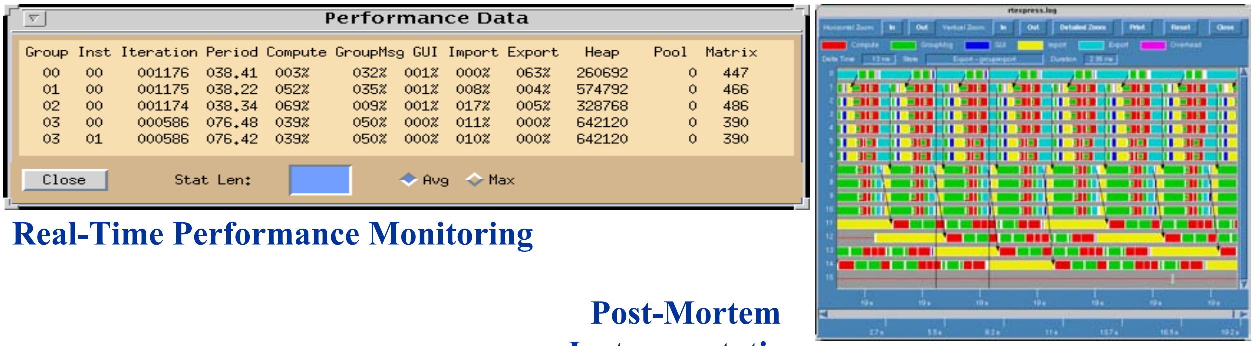
Emulates MATLAB® "image" with enhancements for real-time operation

**Data Pause/Save to disk for User Analysis**
**Variable scaling**
**Time/Space Decimation to control I/O Loading**

**Plan Position Indication (PPI)**

**Figure 36 - H-RTExpress Data Visualization**

The H-RTExpress™ environment provides real-time performance monitoring, shown in Figure 37, as the application is executing on the target architecture, plus post-mortem instrumentation (for debugging and analyzing the parallel application after it has been executed). Although not shown in Figure 37, graphic visualization is supplied for real-time program memory utilization, and real-time program event monitoring.



| Group | Inst | Iteration | Period | Compute | GroupMsg | GUI | Import | Export | Heap | Pool | Matrix |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | 001176 | 038.41 | 003% | 032% | 001% | 000% | 063% | 260692 | 0 | 447 |
| 01 | 00 | 001175 | 038.22 | 052% | 035% | 001% | 008% | 004% | 574792 | 0 | 466 |
| 02 | 00 | 001174 | 038.34 | 069% | 009% | 001% | 017% | 005% | 328768 | 0 | 486 |
| 03 | 00 | 000586 | 076.48 | 039% | 050% | 000% | 011% | 000% | 642120 | 0 | 390 |
| 03 | 01 | 000586 | 076.42 | 039% | 050% | 000% | 010% | 000% | 642120 | 0 | 390 |

**Real-Time Performance Monitoring**

**Post-Mortem Instrumentation**

**Figure 37 - H-RTExpress Performance Monitoring**

### 7.1.1 H-RTExpress™ FPGA Model

The FPGA is viewed as an adjunct to a general-purpose processor (GPP) by H-RTExpress™, as shown in Figure 38. In fact, both Annapolis Micro Systems and the USU/ISI SLAAC group view the FPGA in the same manner. The GPP associated with an FPGA in the H-RTExpress™ system communicates to the other processors via the Message Passing Interface (MPI) standard, and is responsible for all data conversion and control to and from the FPGA logic. H-RTExpress™ allows the programmer to select a function from the FPGA library to be called from the MATLAB language, and an FPGA library element consists of the FPGA Wrapper (C program) and FPGA Core, or set of cores, for the FPGA devices on the WILDSTAR or SLAAC2 boards. The FPGA cores are built and tested for specific devices on the WILDSTAR boards.

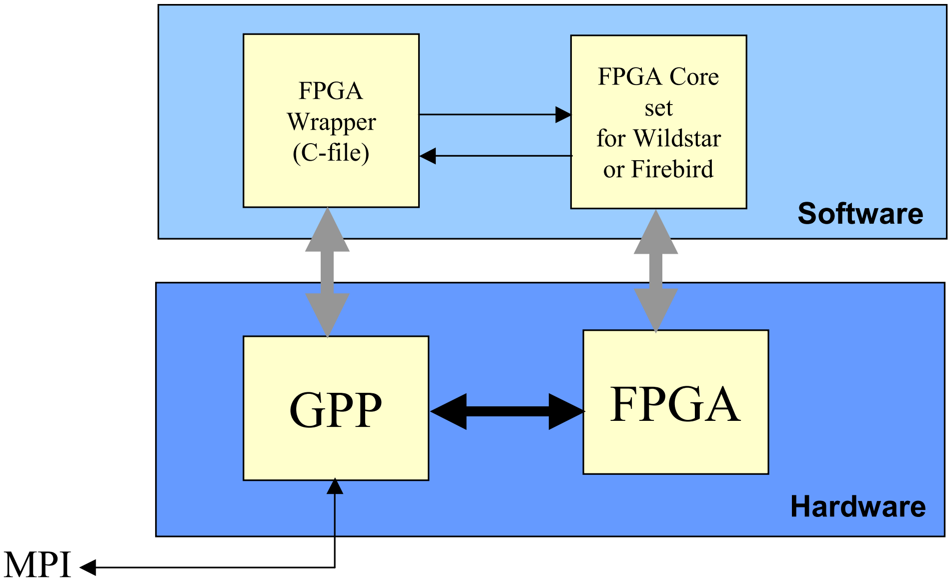
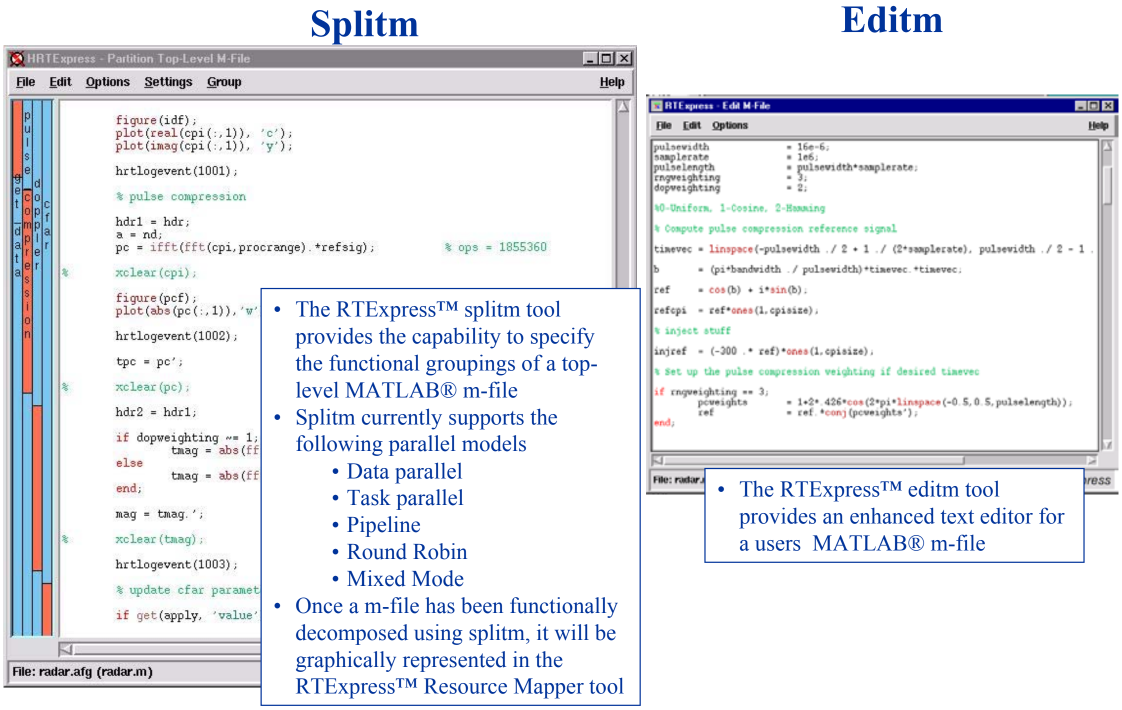


**Figure 38 - H-RTExpress FPGA Model**

### 7.1.2 H-RTExpress™ Tools

The H-RTExpress™ programmer may start with a top-level MATLAB m-file, and using "splitm" take the m-file and split it into logical processing groups. Splitm allows the programmer to define the various groups, and to graphically select lines from the m-file and associate them with the defined groups. The splitm tool creates separate m-files for each processing group. The "editm" tool is very much like splitm, however, it is only concerned with editing m-files, and like splitm, has color-coded presentation to highlight aspects of the MATLAB language. Built-in help for MATLAB and H-RTExpress™ Real-Time Toolbox functions is included. Figure 39 shows the H-RTExpress tools, splitm and editm, screen displays.



Splitm

```
        figure(idf);
        plot(real(cpi(:,1)), 'c');
        plot(imag(cpi(:,1)), 'y');

        hrtlogevent(1001);

        % pulse compression

        hdr1 = hdr;
        a = nd;
        pc = ifft(fft(cpi,procrange).*refsig);          % ops = 1855360

%       xclear(cpi);

        figure(pcf);
        plot(abs(pc(:,1)),'w'

        hrtlogevent(1002);

        tpc = pc';

%       xclear(pc);

        hdr2 = hdr1;

        if dopweighting ~= 1;
                tmag = abs(ff
        else
                tmag = abs(ff
        end;

        mag = tmag.';

%       xclear(tmag);

        hrtlogevent(1003);

        % update cfar paramet

        if get(apply, 'value'
```

File: radar.afg (radar.m)

- The RTExpress™ splitm tool provides the capability to specify the functional groupings of a top-level MATLAB® m-file
- Splitm currently supports the following parallel models
  - Data parallel
  - Task parallel
  - Pipeline
  - Round Robin
  - Mixed Mode
- Once a m-file has been functionally decomposed using splitm, it will be graphically represented in the RTExpress™ Resource Mapper tool

Editm

```
pulsewidth              = 16e-6;
samplerate              = 1e6;
pulselength             = pulsewidth*samplerate;
rngweighting            = 3;
dopweighting            = 2;

%0-Uniform, 1-Cosine, 2-Hamming

% Compute pulse compression reference signal

timevec = linspace(-pulsewidth ./ 2 + 1 ./ (2*samplerate), pulsewidth ./ 2 - 1 .

b       = (pi*bandwidth ./ pulsewidth)*timevec.*timevec;

ref     = cos(b) + i*sin(b);

refcpi  = ref*ones(1,cpisize);

% inject stuff

injref  = (-300 .* ref)*ones(1,cpisize);

% Set up the pulse compression weighting if desired timevec

if rngweighting == 3;
        pcweights       = 1+2*.426*cos(2*pi*linspace(-0.5,0.5,pulselength));
        ref             = ref.*conj(pcweights');
end;
```

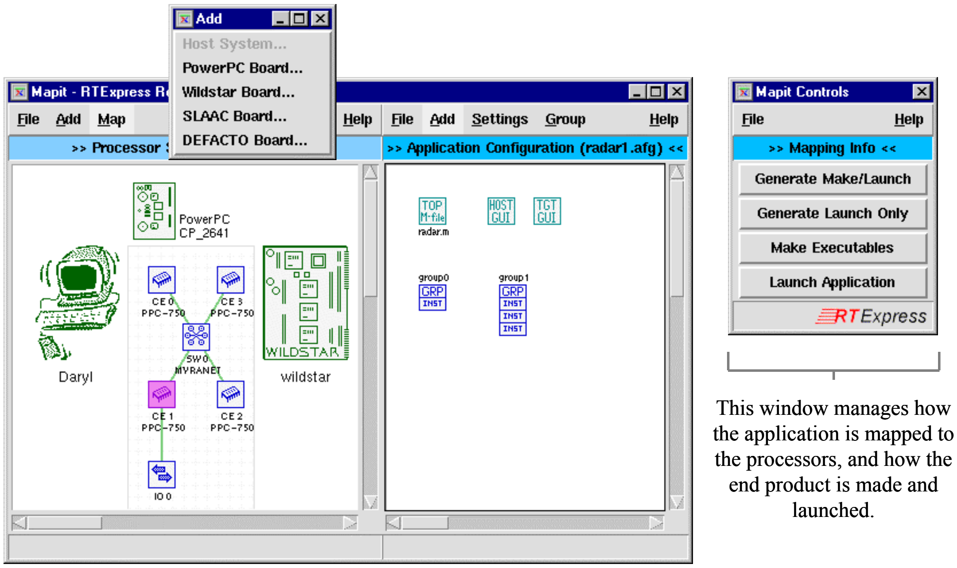
File: radar.

ress

- The RTExpress™ editm tool provides an enhanced text editor for a users  MATLAB® m-file

**Figure 39 - RTExpress Tools - splitm and editm**

Using the H-RTExpress™ Mapit Resource Mapping tool shown in Figure 40, the programmer makes a relationship between processing groups, previously defined using splitm, and processing resources, such as a general-purpose processor or an FPGA board.  A mapping configuration file holds information on the application configuration and processor configuration files and the mapping relations between them.  The application configuration contains information about the top-level m-file, any m-files called, or any C-program files called.  If the processing group is assigned to an FPGA board, the application configuration also contains information about the FPGA core to load and initialize.  The tool produces the standard "make" files and a startup script to launch the parallel program.



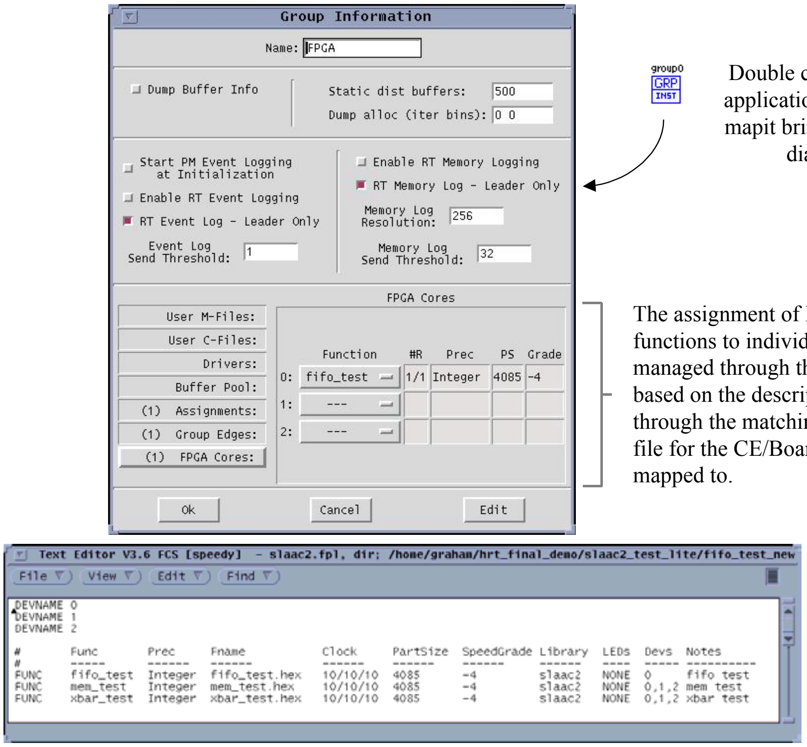
This window manages how the application is mapped to the processors, and how the end product is made and launched.

This side defines what hardware will participate in the implementation (the processor setup)

This side defines how the algorithm is to be broken up for mapping (the application configuration)

**Figure 40 - H-RTExpress Resource Mapping**

If the process group was to be assigned to an FPGA, double clicking its icon (see Figure 41) brings up the FPGA library pop-up menu for specification of an existing FPGA function. The library is extensible, and for this project, the FOPEN SAR Preprocessor core was inserted.



Name: FPGA

☐ Dump Buffer Info

Static dist buffers: 500

Dump alloc (iter bins): 0 0

☐ Start PM Event Logging at Initialization

☐ Enable RT Event Logging

☒ RT Event Log - Leader Only

Event Log Send Threshold: 1

☐ Enable RT Memory Logging

☒ RT Memory Log - Leader Only

Memory Log Resolution: 256

Memory Log Send Threshold: 32

FPGA Cores

| | Function | #R | Prec | PS | Grade |
|---|---|---|---|---|---|
| 0: | fifo_test | 1/1 | Integer | 4085 | -4 |
| 1: | --- | | | | |
| 2: | --- | | | | |

```
DEVNAME 0
DEVNAME 1
DEVNAME 2

#       Func       Prec     Fname            Clock     PartSize  SpeedGrade Library  LEDs  Devs   Notes
#       -----      ------   ------           ------    ------    ------     ------   ----  -----  ----------
FUNC    fifo_test  Integer  fifo_test.hex    10/10/10  4085      -4         slaac2   NONE  0      fifo test
FUNC    mem_test   Integer  mem_test.hex     10/10/10  4085      -4         slaac2   NONE  0,1,2  mem test
FUNC    xbar_test  Integer  xbar_test.hex    10/10/10  4085      -4         slaac2   NONE  0,1,2  xbar test
```

Double clicking an application group in mapit brings up this dialog

The assignment of FPGA core functions to individual devices is managed through this interface, based on the descriptions provided through the matching FPGA library file for the CE/Board the group was mapped to.

**FPGA Library Description File (SLAAC2)**

**Figure 41 - H-RTExpress FPGA Library Selection**

### 7.1.3 H-RTExpress™ Extendibility

H-RTExpress™ allows for programmers to begin with a single top-level MATLAB m-file, several m-files, m-files, and m-files that are called as functions. C programs may be used with H-RTExpress™ as User-C functions, and as previously discussed, FPGA cores can also be put to use. The FPGA core library is user extendible, and each library element consists of a C-wrapper file and one or more FPGA cores that are associated with it. The cores are built and tested for a specific function and specific FPGA hardware.

## 7.2 COREFIRE

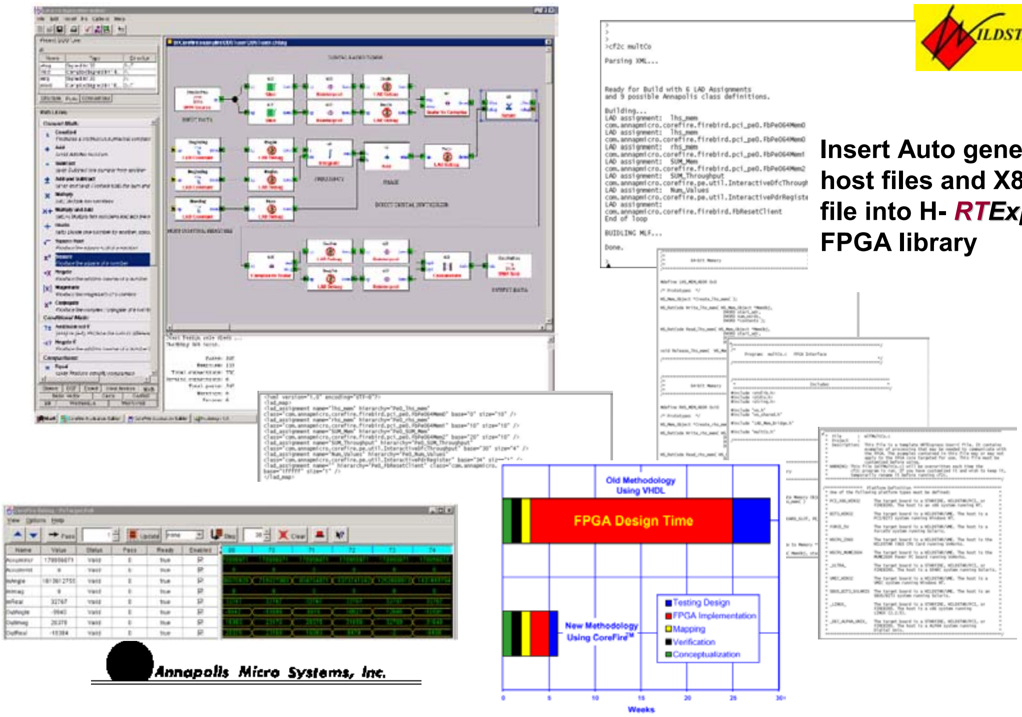
Annapolis Microsystems was called on to create the FPGA images for testing under this program. Rather than follow the standard and lengthy VHDL development, use of the newly created COREFIRE design tool was selected. The benefits of this design suite included very rapid development of high performance applications, and the ability for those applications to be created and modified by engineers at ISI without additional time required from Annapolis Microsystems.

COREFIRE succeeded in cutting the FPGA development time allowing investigation of several cores, and also allowing completion of the Software Radio Correlate function within schedule constraints.

The COREFIRE ™ Design Suite provides a large collection of cores and an application builder that enable users to describe and build FPGA designs without using hardware design languages such as VHDL or Verilog. The COREFIRE ™ Design Suite also includes an application debugger that enables users to load and interact with their designs in the target environment without writing application programs. As the central component of the COREFIRE ™ Design Suite, the COREFIRE ™ Application Builder features a unique drag-and-drop capability for combining core components into ready-to-run programs. See Figure 42 for an overview of the COREFIRE environment. An extensive array of these core components are available in the COREFIRE ™ Module Libraries, along with Annapolis Micro Systems, Inc. board-specific support packages.

In summary, some of the benefits of COREFIRE include:

- Works from High Level, Data Flow Concept of the Design
- Combines GUI Design Entry and Debug Tools with Tested, Optimized CoreFireTM IP Cores
- Drag and Drop High and Low Level Modules
- CoreFireTM Modules Incorporate Years of Application Development Experience - Highly Optimized and Tested
- CoreFireTM Tools and Modules are Intelligent
- Modules Automatically Handle Synchronization
- Manage Clocks and Other Low Level Hardware Signals
- Guarantee Correct Control by Design
- Modules "Know How" to Interact With Each Other
- Board Support Packages Incorporate Hardware Details of the Boards - Invisible to Users
- Supports Conversion Between Data Types - Bit, Signed and Unsigned Integers, Single Precision Floating Point, and Integer and Floating Point Complex Data Types
- Provides Java Files & Host Code to Run & Debug the FPGAs
- Works with all WILDTM VirtexTM, VirtexTM E and VirtexTM II FPGA processor and I/O boards



WILDSTAR™
Insert Auto generated C host files and X86 image file into H- RTExpress™ FPGA library
Old Methodology Using VHDL
FPGA Design Time
New Methodology Using CoreFire™
■ Testing Design
■ FPGA Implementation
■ Mapping
■ Verification
■ Conceptualization
0
5
10
15
20
25
30
Weeks
Annapolis Micro Systems, Inc.

```
>
>
>
>cf2c multCo

Parsing XML...


Ready for Build with 6 LAD Assignments
and 9 possible Annapolis class definitions.

Building...
LAD assignment:  lhs_mem
com.annapmicro.corefire.firebird.pci_pe0.FbPe064Mem0
LAD assignment:  lhs_mem
com.annapmicro.corefire.firebird.pci_pe0.FbPe064Mem0
LAD assignment:  rhs_mem
com.annapmicro.corefire.firebird.pci_pe0.FbPe064Mem1
LAD assignment:  SUM_Mem
com.annapmicro.corefire.firebird.pci_pe0.FbPe064Mem2
LAD assignment:  SUM_Throughput
com.annapmicro.corefire.pe.util.InteractiveDfcThrough
LAD assignment:  Num_Values
com.annapmicro.corefire.pe.util.InteractivePdrRegiste
LAD assignment:
com.annapmicro.corefire.firebird.FbResetClient
End of loop

BUIDLING MLF...

Done.

>
```

**Figure 42 - Annapolis Microsystems COREFIRE Development Environment**

An ISI generated utility reads the XML map file from COREFIRE and create a series of files to facilitate insertion of the newly generated FPGA core into the H-RTExpress core library. The basic operation of Corefire to C (cf2c) is to read the XML file of host interface definitions, picking up the user's name and address for each item and the Corefire name for the item. Based on the Corefire name, the appropriate template is selected. That template is appended to the *.c file (lower-case name) and its associated *.h file (lower-case-name). The template contains various key words that become replaced with the user's interface item name and local address. The mlf*.c file becomes the C-wrapper that is a callable function from the RTExpress MATLAB, and it uses the interface functions that were created by this process. The user must edit the main portion of the mlf file to produce the desired behavior, however, all of the procedures to support each interface item have been created to expedite the process of creating the wrapper file. The new FPGA function is then entered into the FPGA Library Description File for use by H-RTExpress.

# 8 Experiments and Collected Data

## 8.1 Initial Demonstration

An initial demonstration was performed to verify the performance of the FIREBIRD hardware within the PC chassis. Part of this demonstration was to also verify operation of the software radio code that was received from AFRL.

To examine whether the instrumentation and interface with H-RTExpress, a small example program was constructed. A task for the FPGA was created to shift alternating zeros and ones down a very long shift register with the purpose of maximizing the toggle rate within the FPGA. Figure 43 shows the VHDL used for the simple "heater-core" program. The sole purpose of the program was to draw power. The test was run at various frequencies, and the FIREBIRD current and voltage sensors were polled to collect regular samples of power used. With the help of H-RTExpress, the results were displayed in time on the screen.

```
-----------------------------------------------------------------------------
--  shift data
-----------------------------------------------------------------------------

-- purpose : shifter
-- inputs  : user clock, reset
-- outputs : none
shifter_p : process (Clocks_In.U_Clk, Global_Reset)
  variable shift_data : bit_vector(10000 downto 0) := (others => '0');
  variable xx         : bit := '0';

begin  -- process shifter_p
  if Global_Reset = '1' then              -- asynchronous reset
    shift_data := (others =>'0');
    shift_data(0) := '1';
    LEDs_Out.Green_n   <= '0';
    LEDs_Out.Yellow_n  <= '0';
    LEDs_Out.Red_n     <= '0';

  elsif Rising_Edge(Clocks_In.U_Clk) then  -- rising clock edge

    xx := not shift_data(0);

    if xx = '0' then
      LEDs_Out.Red_n     <= '0';
    else
      LEDs_Out.Red_n     <= '1';
    end if;

    shift_data(shift_data'length-1 downto 1) := shift_data(shift_data'length-2 downto 0);
    shift_data(0) := xx;
  end if;
end process shifter_p;
```

**Figure 43 - Shift Register VHDL**

As expected, when the clock frequency to the FPGA was increased, power consumption also increased. This was seen immediately in the on-screen plot and in a text window as shown in Figure 44. The current and voltage values were recorded and also plotted versus frequency as shown in Figure 45.

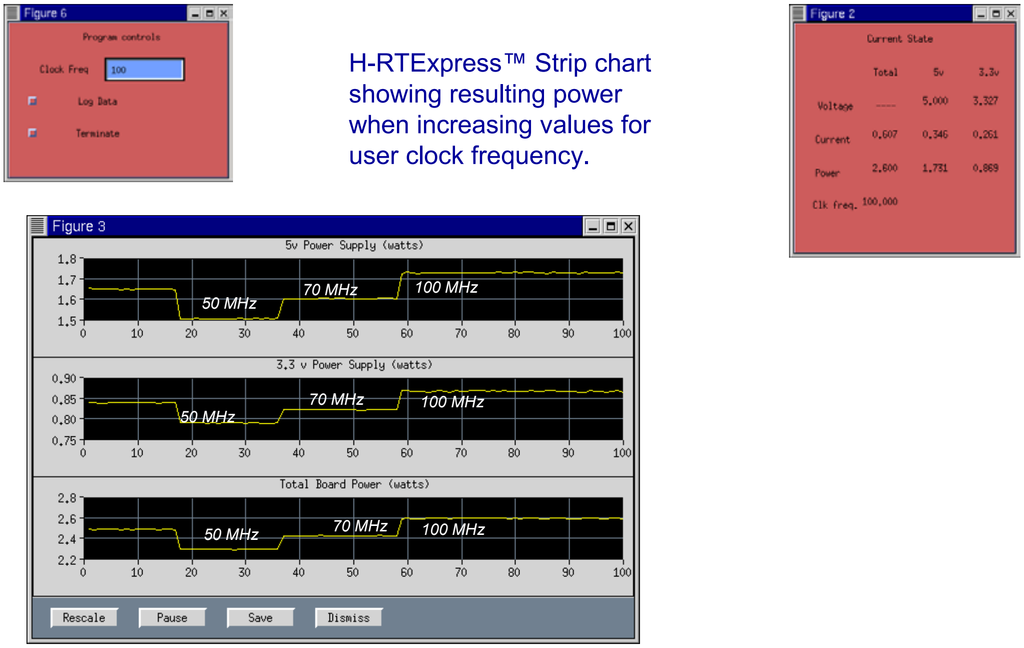


Figure 6

Program controls

Clock Freq 100

□ Log Data

□ Terminate

H-RTExpress™ Strip chart showing resulting power when increasing values for user clock frequency.

Figure 2

Current State

|  | Total | 5v | 3.3v |
|---|---|---|---|
| Voltage | ----- | 5.000 | 3.327 |
| Current | 0.607 | 0.346 | 0.261 |
| Power | 2.600 | 1.731 | 0.869 |
| Clk freq. | 100.000 |  |  |

**Figure 44 - Initial Demo - HRTExpress Display**

The FPGA core power use curve versus FPGA clock frequency was compared to a predicted curve that was produced using the Xilinx Power Estimation Spreadsheet tool. The Spreadsheet tool requires user input to place FPGA part utilization numbers from the Xilinx map file into the spreadsheet, and also the user must estimate the toggle-rate of the design. The toggle rate figure is dependent not only on the design but the nature of the data to be processed, and general rule of thumb guidelines are provided with the spreadsheet directions. For this test, the toggle rate was deliberately maximized, however, in real world problems, it is difficult to know exactly what the toggle rate is.

## FIREBIRD™ Power Measurement

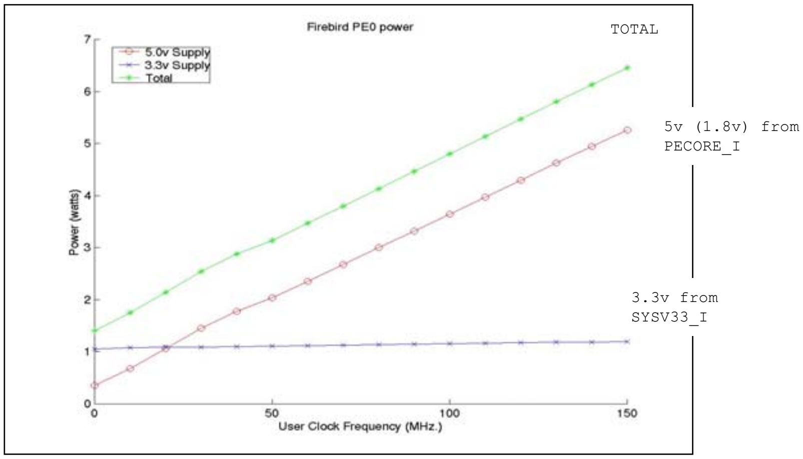
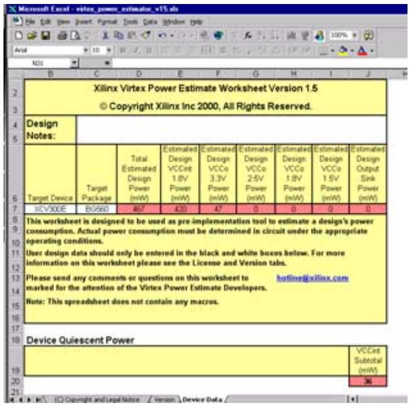


**Figure 45 - Shift Register Example - Power vs Frequency**

As shown in Figure 46, correlation with the power estimator was very good for this example. This is largely due to the high toggle rate of this synthetic problem.



Microsoft Excel - virtex_power_estimator_v15.xls

Xilinx Virtex Power Estimate Worksheet Version 1.5

© Copyright Xilinx Inc 2000, All Rights Reserved.

Design Notes:

| Target Device | Target Package | Total Estimated Design Power (mW) | Estimated Design VCCint 1.8V Power (mW) | Estimated Design VCCo 3.3V Power (mW) | Estimated Design VCCo 2.5V Power (mW) | Estimated Design VCCo 1.8V Power (mW) | Estimated Design VCCo 1.5V Power (mW) | Estimated Design Output Sink Power (mW) |
|---|---|---|---|---|---|---|---|---|
| XCV300E | BG560 | 467 | 420 | 47 | 0 | 0 | 0 | 0 |

This worksheet is designed to be used as pre implementation tool to estimate a design's power consumption. Actual power consumption must be determined in circuit under the appropriate operating conditions.

User design data should only be entered in the black and white boxes below. For more information on this worksheet please see the License and Version tabs.

Please send any comments or questions on this worksheet to hotline@xilinx.com marked for the attention of the Virtex Power Estimate Developers.

Note: This spreadsheet does not contain any macros.

Device Quiescent Power

| VCCint Subtotal (mW) |
|---|
| 36 |

| Frequency | Target Device | Target Package | Total Estimated Design Power (mW) | Estimated Design VCCint 1.8V Power (mW) | Estimated Design VCCo 3.3V Power (mW) |
|---|---|---|---|---|---|
| 0 | XCV2000E | FG680 | 270 | 270 | 0 |
| 10 | XCV2000E | FG680 | 729 | 682 | 47 |
| 20 | XCV2000E | FG680 | 1189 | 1095 | 94 |
| 30 | XCV2000E | FG680 | 1649 | 1508 | 141 |
| 40 | XCV2000E | FG680 | 2110 | 1922 | 188 |
| 50 | XCV2000E | FG680 | 2570 | 2335 | 235 |
| 60 | XCV2000E | FG680 | 3030 | 2748 | 282 |
| 70 | XCV2000E | FG680 | 3490 | 3161 | 329 |
| 80 | XCV2000E | FG680 | 3952 | 3575 | 377 |
| 90 | XCV2000E | FG680 | 4412 | 3988 | 424 |
| 100 | XCV2000E | FG680 | 4872 | 4401 | 471 |
| 110 | XCV2000E | FG680 | 5332 | 4814 | 518 |
| 120 | XCV2000E | FG680 | 5793 | 5228 | 565 |
| 130 | XCV2000E | FG680 | 6253 | 5641 | 612 |
| 140 | XCV2000E | FG680 | 6713 | 6054 | 659 |
| 150 | XCV2000E | FG680 | 7174 | 6467 | 707 |

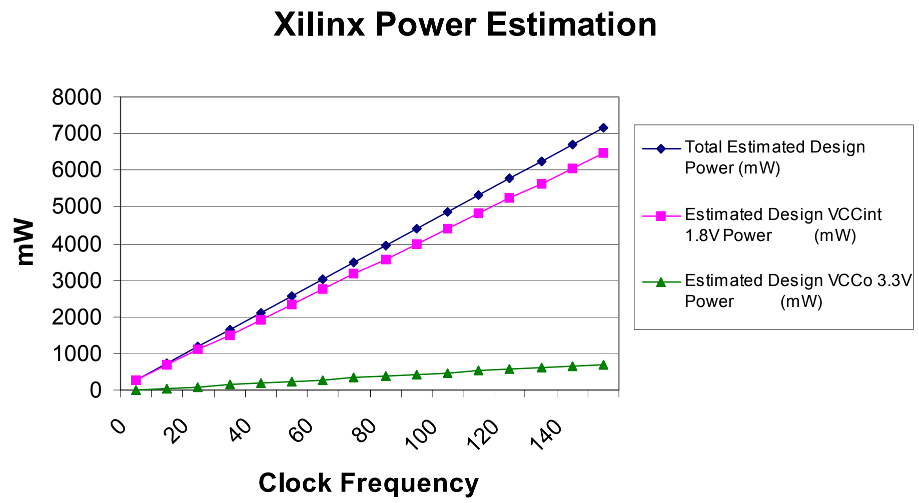


**Figure 46 - Xilinx Power Estimation Spreadsheet**

To gain experience with the COREFIRE and H-RTExpress tools and operation of the FIREBIRD board, several small tests were constructed using simple functions. The nature of the tests was pretty much the same. Two vectors of numbers were loaded into two separate memories of the FIREBIRD board, and then the operation was applied with the result stored in another memory of the FIREBIRD board. Timing was accomplished using the COREFIRE "throughput" register. A "simplified" block diagram of the circuit for the Throughput Register is shown in Figure 47.

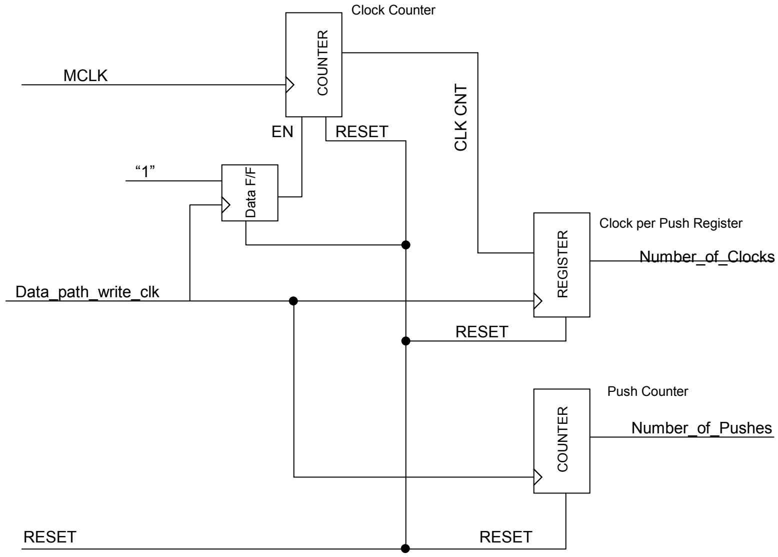


**Figure 47 - COREFIRE Throughput Register**

The throughput register supplies two counts, the number of data items pushed through the path, and the number of clocks per push.

The input memories are accessed through the use of counters, which need to know the number of elements in the vector to count up to. Since COREFIRE is a dataflow diagram-programming tool, each COREFIRE function is executed when data is valid, or ready, at all of its inputs. Once a data item is used, it is consumed and no longer available. COREFIRE does provide a function to continuously echo a data item, if needed. Also, host registers come in two varieties. There is the normal host register, which is loaded with a value from the host, and then that value is consumed by the dataflow diagram. There is also a host Constant register, which is also loaded from the host, however, its value is continuously available to the dataflow diagram. For power measurements, host constant registers were used, which caused the sample operation to run indefinitely. Since the inputs were static in memory, the FPGA board would be in an infinite loop and polling the voltage and current points by software from the host was reasonable. If the operation only ran one time, it would be too difficult, given the short execution time of the FPGA for the host I/O to fully catch events power characteristics.

Test functions were built with increasing complexity on both GPP and FPGA for timing and power measurements and comparison. The functions included adding 24-bit integers, 32-bit integers, 32-bit floating point, and complex floating point (32-bit real / 32-bit imaginary). Multiplication with 24-bit integers, 32-bit integers, 32-bit floating point, and complex floating point (32-bit real / 32-bit imaginary) was done, as was a Quadratic Solver, a complex floating point FFT, and the Correlate Function for the Software Radio.

## 8.2 Add

The two dataflow paths, as seen in **Error! Reference source not found.**, leading up to the input of the add block are identical. The host computer loads the "lhs_mem" and "rhs_mem" with two collections of numbers that are to be added. The "iterate end" block is a counter that counts from zero to the number specified by the host in the "LAD Constant" block. The constant register continuously supplies that end count to the diagram so that the counters will repeat accessing the block of memory containing the input vectors, and the add diagram will operate indefinitely. The "read" block presents the memory contents, and in the example shown, the 64-bit

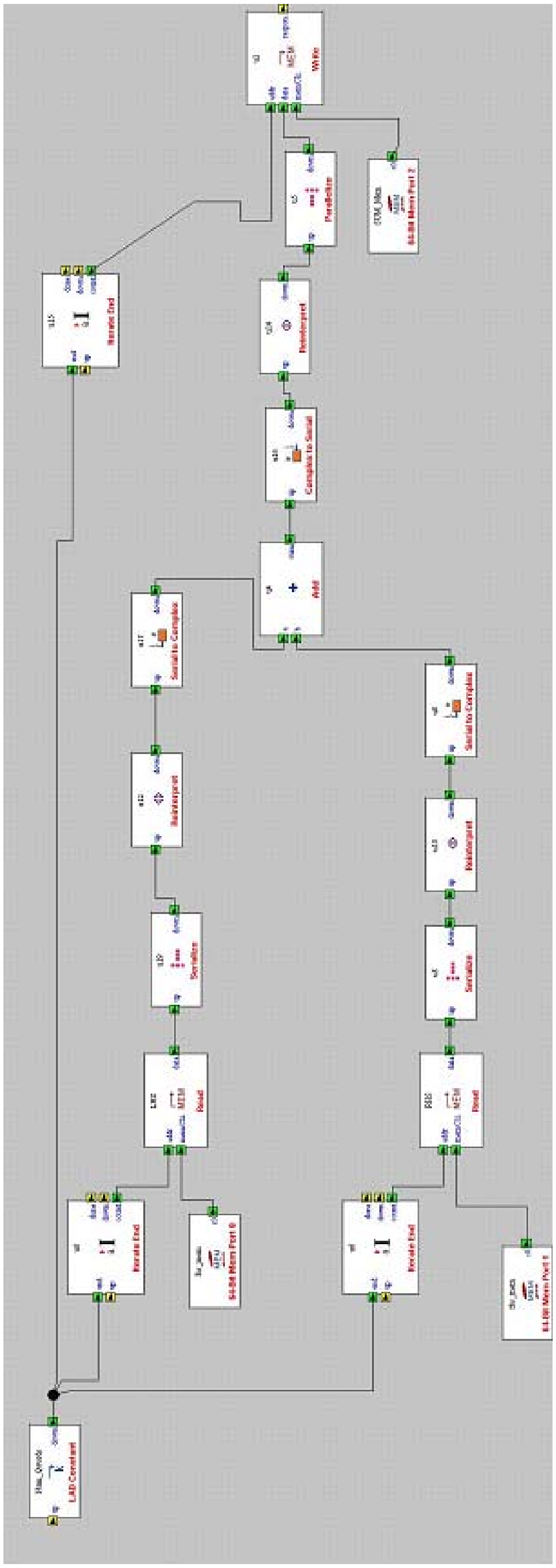
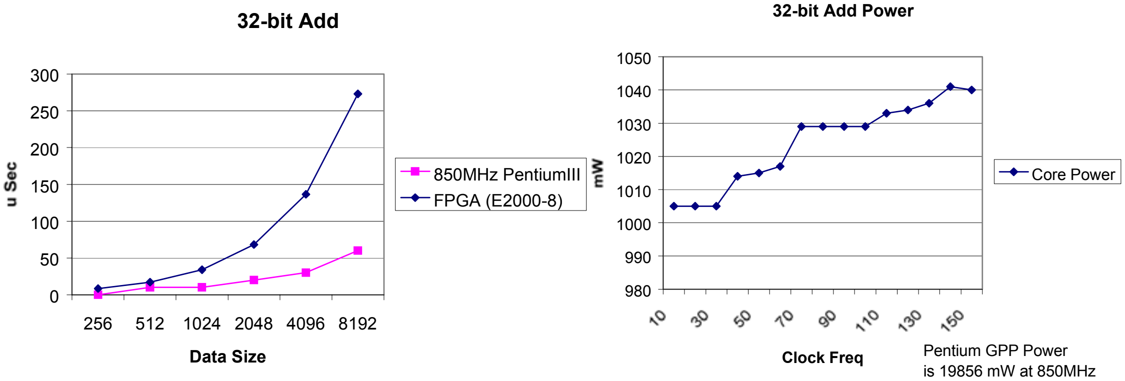


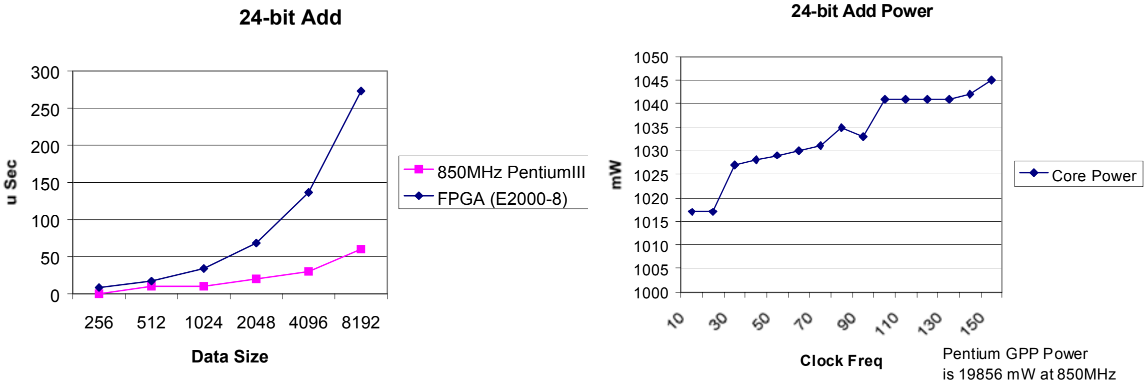
Figure 48  COREFIRE "add" Dataflow Example

memory has been loaded with two 32-bit floating point numbers in each memory location. The "serialize" block has been set up to break the two halves of the 64-bit word and send each 32-bit number down the path. All information in the memory is treated as "bits" however, the "Interpret" block is used to let downstream dataflow functions know that the bits are to be interpreted as floating point numbers. The next step is "serial to complex" which takes the numbers as pairs of alternating real and complex values and converts the path to that of the complex data type. The adder performs the operation consistent with the data presented to it, so a complex add is completed.



Pentium GPP Power
is 19856 mW at 850MHz

**Figure 49 - 32-bit Integer Add**



Pentium GPP Power
is 19856 mW at 850MHz

**Figure 50 - 24-bit Integer Add**

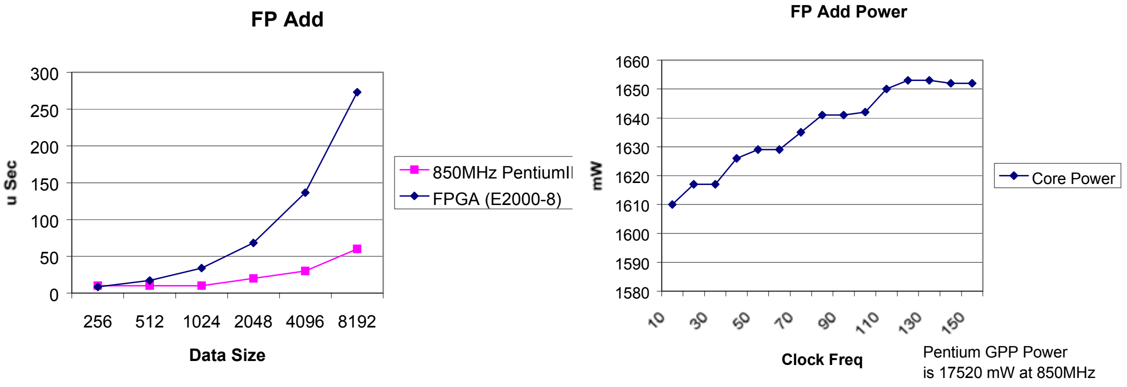


**Figure 51 - 32-bit Floating Point Add Time**

The charts above, Figure 49, Figure 50, Figure 51, show the measurements taken for the various add tests. Of course, the FPGA running at 100 MHz doing a serial add compared to the same process on an 850 MHz Pentium computer is going to be slower. However the comparison of the power trend in each example shows that increasing clock frequency increases the power consumed on the FPGA while a constant power was measured on the general-purpose processor. No facility existed to modify the clock frequency for the general-purpose processor.

## 8.3 Mult

A similar set of measurements was taken using multiplication. The dataflow diagram for this test is similar in structure as the addition tests. Two vectors of data are placed in separate memories and a counter is used to access the data elements in each memory and present them to the multiplier. The 32-bit integer data is packed into the 64-bit memories to take advantage of that width and the bandwidth of the 64-bit PCI interface. The serialize function pulls each 32 bit piece out in sequence. For the complex multiply, the 32 bit halves of the memory were interpreted as floating point numbers, with the real and complex pairs sharing each 64 bit location.

For power measurements, the diagrams were run continuously, however, timing measurements using the throughput registers were made with a single run.

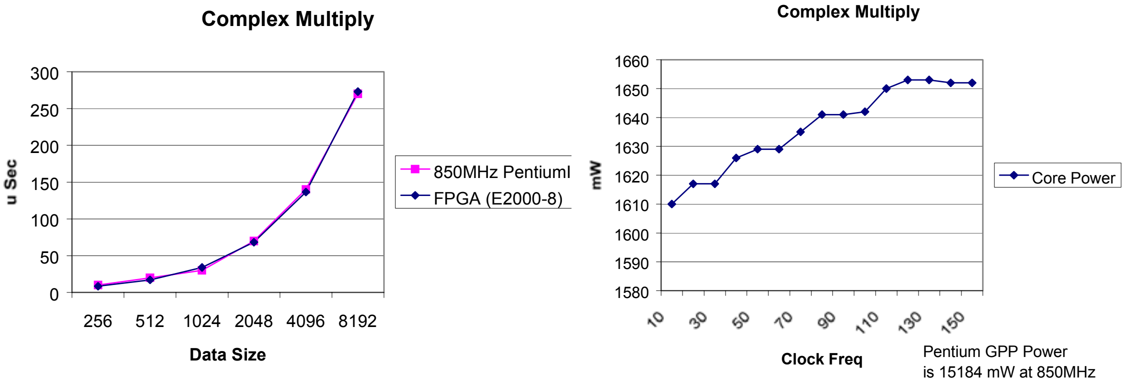


**Figure 52 - 32-bit Floating Point, Complex Multiply Time**

The measured results for the 32-bit floating-point complex multiply are shown in Figure 52, and the results follow the same trend with increasing power consumption as operating frequency increases. Compared to the general-purpose processor (GPP), the FPGA is nearly equivalent in performance, however the GPP power consumption is constant. There was no mechanism to alter the clock frequency on the GPP.

## 8.4 FFT

The following times were collected for the throughput registers in the FFT test diagram shown in Figure 53, built with COREFIRE, and run on the FIREBIRD with FPDP card. Both the baseboard and I/O board had Xilinx Virtex E 2000-8 parts. The test was run at 100 MHz. The base-board DMA channel was used, and a Java interface program exercised the function.

The present COREFIRE tool has an FFT core with a different interface than was available when this test was run. The FFT had a "token" interface such that a four-bit value, or token, was presented with the start of an input spectrum. Additional spectrums could be loaded into input buffers for other token values, but since the FFT was an "in-place" operation, the input for a token could not be over written until the output spectrum completed. On the last data item for an output spectrum, the token value for that data set is output.

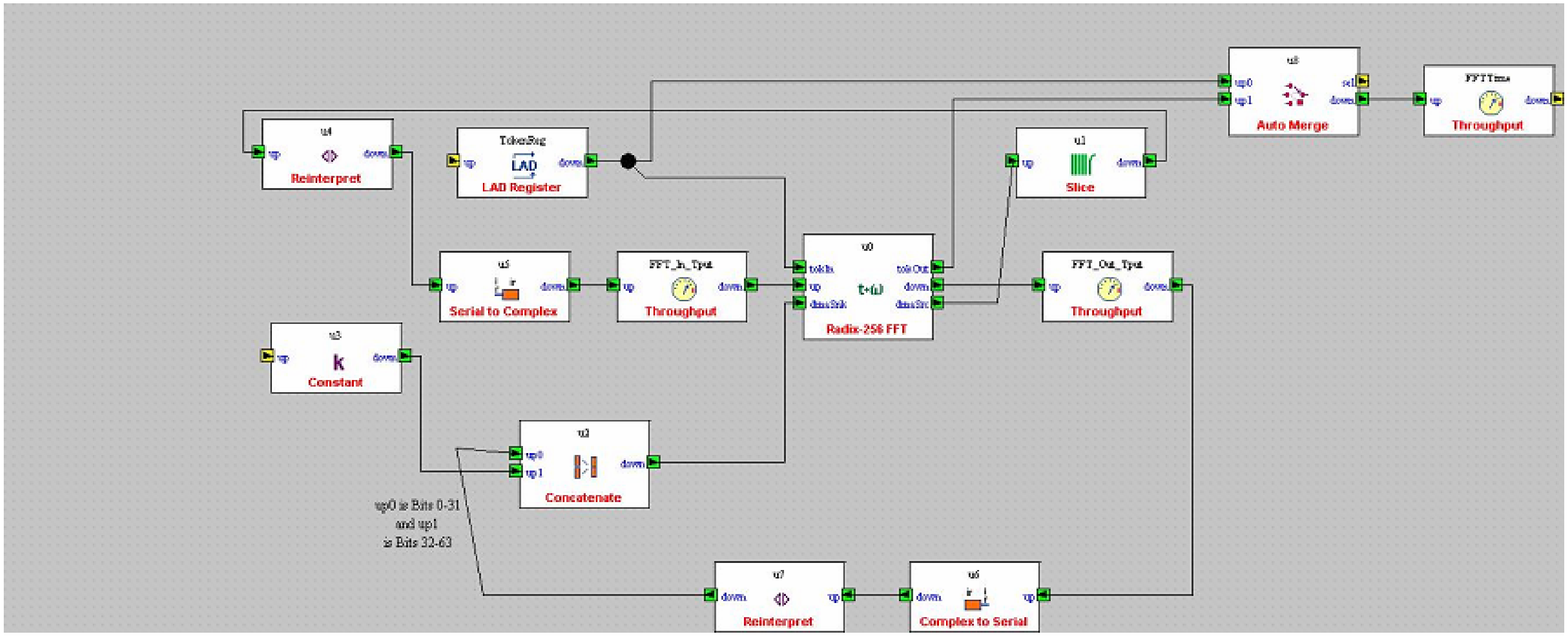
Taking advantage of the operation of this FFT implementation the DMA write was initiated to present data to the FFT, and the DMA read was also started to allow the output path. The last item written was the TokenReg value into the LAD Register, and at that point the FFT core had every input available and would fire. [Note that the dma sink and dma source on the FFT block in this diagram is actually a short-cut path to the baseboard dma channel. The FFT core actually resides on the baseboard, while the remainder of this diagram is resident on the i/o board FPGA.]

Starting from the dma source, which is a 64-bit width path, the data is sliced to use only the lower 32 bits. A reinterpret is used for the downstream blocks to interpret the 32 bits as floating point, and a serial to complex is used to interpret pairs of the 32-bit floating point values as complex floating point real and imaginery pairs. A throughput register at the input of the FFT counts the data items and the clocks.

On the output of the FFT, another throughput register counts the data items pushed through and the clocks. The complex pairs are returned to a serial stream of 32-bit floating point values and concatenated with 32 bits of zeros in the upper half of the resulting 64-bit word. The 64-bit result is then sent to the dma sink to be returned to the host computer.

As noted earlier, the process is kicked-off with the write of the LAD Register that holds the token value. The output of the register is fed to the FFT block, and also to an auto merge that feeds another token register. Another input of the auto merge is the token out of the FFT block, which should be output on the last data value of the output spectrum. Therefore, the throughput register should record two pushes, one for the writing of the token to start the FFT and one for the output of the token signaling the end of the FFT processing. The number of clocks recorded in this throughput register then are the total number of clocks for the process.

The table shown in Figure 53 shows, for each size of spectrum tested, the Total clocks recorded by the throughput register associated with the auto merge. Each line shows 2 pushes for the two tokens and the number of clocks at the recorded clock frequency of 100 MHz. Using the throughput registers at the input and output of the FFT, which are measuring the data transfer clocks in and out of the FFT buffers to and from dma, and subtracting that number of clocks from the total number of clocks, gives the clock cycles for the FFT processing. Multiplying the period of the clock programmed, FFT time in microseconds is listed. For example, a 1K complex-floating-point FFT takes 31 microseconds at 100 MHz.



| | | | Total | | In | | Out | | microseconds | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Part | Clock(MHz) | Size | Push | Clocks | Push | Clocks | Push | Clocks | FFT Time | Data Input | Data Output | Total |
| Xilinx 2000E-8 | 100 | 256 | 2 | 121105 | 241 | 1797 | 256 | 118308 | 10.000 | 17.97 | 1183.08 | 1211.05 |
| Xilinx 2000E-8 | 100 | 512 | 2 | 123215 | 497 | 4341 | 512 | 116778 | 20.960 | 43.41 | 1167.78 | 1232.15 |
| Xilinx 2000E-8 | 100 | 1024 | 2 | 126182 | 1009 | 9526 | 1024 | 113533 | 31.230 | 95.26 | 1135.33 | 1261.82 |
| Xilinx 2000E-8 | 100 | 2048 | 2 | 135285 | 2033 | 19936 | 2048 | 110007 | 53.420 | 199.36 | 1100.07 | 1352.85 |
| Xilinx 2000E-8 | 100 | 4096 | 2 | 142866 | 4081 | 40813 | 4096 | 92714 | 93.390 | 408.13 | 927.14 | 1428.66 |
| Xilinx 2000E-8 | 100 | 8192 | 2 | 142806 | 8177 | 81956 | 8192 | 42926 | 179.240 | 819.56 | 429.26 | 1428.06 |

**1K 1D FFT Processing time is 31 micro-sec at 100 MHz**

*Max clock available is 150 MHz

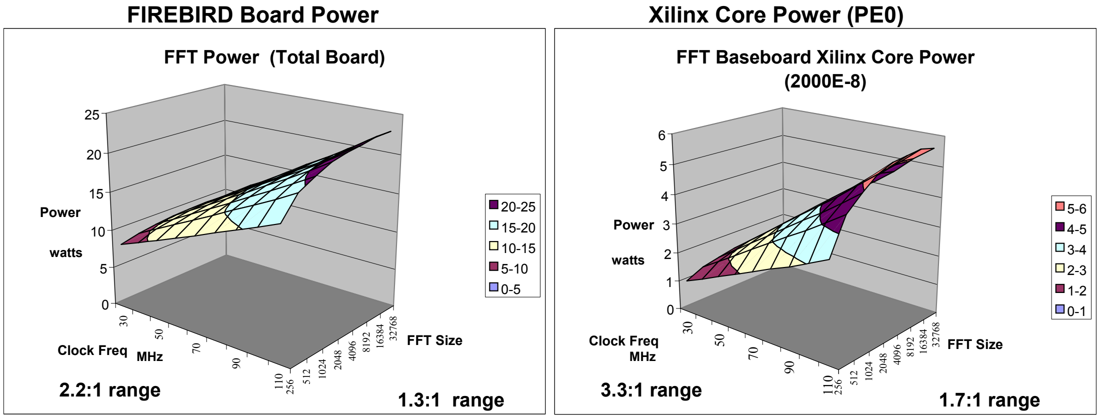
**Figure 53 - FFT Performance Measurement using COREFIRE**

The processing time measurement agreed with the processing times submitted by Annapolis Micro Systems (AMS), lending confidence in the test method at ISI. Next, the FFT was measured for power consumption. As with the previous testing, the FFT was put into continuous use while the fbmon tool logged data taken from the instrumentation points on the FIREBIRD board. The charts in Figure 54 show the total power draw by the board and also the power drawn by the core of the baseboard Xilinx FPGA. The baseboard FPGA executes the AMS FFT core and the I/O board contains the user's diagram, which in this case has minimal logic.

As illustrated in Figure 54 the clock frequency was varied from 30 MHz to 110 MHz, and for each spectrum size, ranging from 256 to 32K, power data was recorded. The AMS core is a radix-256 floating point complex FFT. Varying clock frequency gave on the order of a 2.2 to 1 variation in power, while varying the spectrum size gave on the order of 1.3 to 1 variation in power when looking at the total board power. Just focusing on the Xilinx FPGA running the FFT, turning the clock frequency knob resulted in a 3.3 to 1 power variation, and turning the spectrum size knob gave a 1.7 to 1 variation.

For reference, the baseboard total-board power was measured with a "safe-core" loaded. The "safe-core" has no operating logic and has inputs and outputs tied off. The board power measured ranged from 5 to 6 watts as clock frequency varied from 30 to 110 MHz.

Of interest is the comparison of the FPGA performance to a General Purpose Processor. Unfortunately, the microprocessor motherboard was not instrumented in the same way as the FPGA board, so direct comparison of power values is not valid. Trends in the power variation, however, are visible. For the 850-MHz Intel Pentium III processor, a constant power of 15.4 watts was measured as the spectrum size was varied from 256 to 32K samples. As discussed earlier, the 15.4 watts is the difference in total system power between a continuous running of the FFT example, and a continuous run with no application (idle). Therefore, the 15 watts should represent the delta in power required by the FFT application. Total FPGA board power ranged from 8.5 watts to 22.7 watts, looking at lowest clock and smallest spectrum to highest clock and largest spectrum. The general purpose processor power was constant.


FIREBIRD Board Power

Xilinx Core Power (PE0)

**FPGA Power consumed increases with clock and with FFT Size.**

**GPP Power for same function is 15.4 W at all sizes.**
**No change in power due to spectrum size.**

**Base power of FPGA Board ranges between 5 to 6 watts between 30 to 110 MHz**

**Figure 54 - FFT Power Profile on FIREBIRD**

The general-purpose processor times were measured for each spectrum size and are shown in Figure 55. The FFT times on the FPGA had also been measured.

|  |  |  | mwatt-sec | mwatt-sec |
|---|---|---|---|---|
| size | Watts | sec | Energy | Energy |
| 256 | 15.366 | 0.000019 | 0.29 | 0.16 |
| 512 | 15.366 | 0.000043 | 0.66 | 0.37 |
| 1024 | 15.366 | 0.000097 | 1.49 | 0.57 |
| 2048 | 15.366 | 0.000227 | 3.49 | 1.02 |
| 4096 | 15.366 | 0.00052 | 7.99 | 1.87 |
| 8192 | 15.366 | 0.00118 | 18.13 | 3.65 |
| 16384 | 15.366 | 0.002793 | 42.92 | 7.05 |
| 32768 | 15.366 | 0.007932 | 121.88 | 13.83 |

**Figure 55 - Linux FFT Times on 850 MHz P III**

Factoring in time to process with the power consumed, Figure 56 shows the energy required by the FFT on the FPGA. As before, the two charts in Figure 56 display the total FIREBIRD board energy on the left and just the Xilinx FPGA core energy on the right. Factoring in processing time flatten the variation seen by changing the clock frequency. Since the same work is being done, it is reasonable to expect that the same amount of energy is required to do a specific amount of work whether that work is done fast or slow. The variation in energy required due to spectrum size, however, is now amplified by the amount of time required to process the spectrum size. For the total board, that variation is on the order of 100 to 1, and the Xilinx FPGA running the FFT exhibited about a 90 to 1 range in energy required.

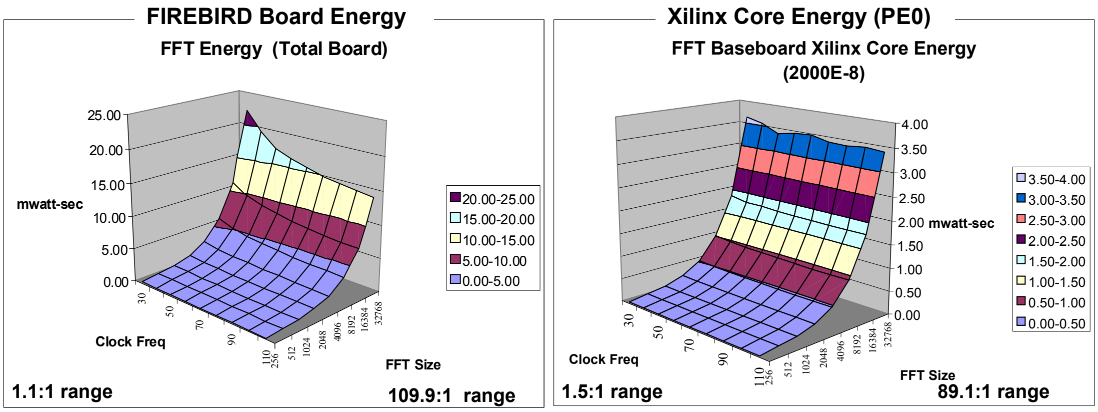


**Figure 56 - FFT Energy Measurement**

The time to serially process the larger spectrum size FFT grows quickly as spectrum size increases on the general-purpose processor as seen in Figure 57.

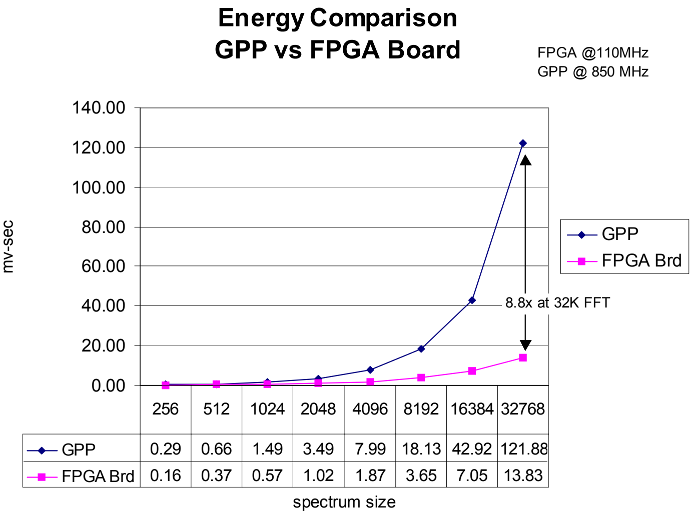


**Figure 57 - FFT Energy Comparison - FPGA vs. GPP**

The benefit of the FPGA implementation can be seen at larger FFT sizes where the serial GPP operation time becomes very long. The variation in energy due to the size of the FFT spectrum, looking at the values for the board's energy, is up to a 30x range between 256 and 8K, or 108x between 256 and 32K.

The devices measured included the Virtex2000E-8 at 100 MHz and the 850 MHz PentiumIII with Linux Operating System.

In the course of taking data measurements on the FFT core, some unexpected results were obtained. The plots of Figure 58 shows the total FIREBIRD board power consumption on the plot in the left of the figure. As frequency increased power generally increased, and as spectrum size increased power consumption generally increased, but the surface is not smooth. Looking at the plot on the right half of the figure, the Xilinx Core power at the FPGA is very strange. At the low end of the frequency dial, there is much more power required than at the high end of the frequency scale. There also appears to be an unexplained variation in power consumption as spectrum size increases. This is certainly not the result that was postulated.

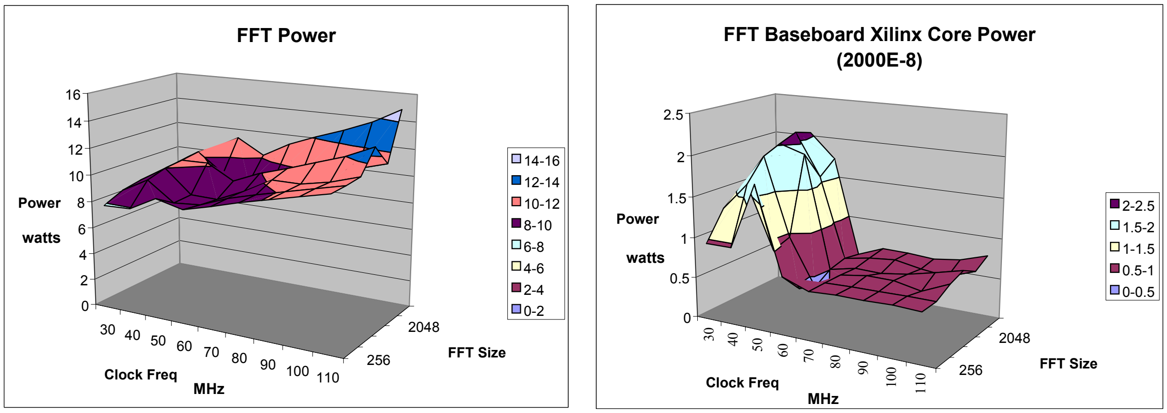


**Figure 58 - Example of not resetting FPGA DLL**

The early technique used at ISI for taking the power measurements was to program the board clock frequency, load the FPGA core, and reset the board. The core would operate as power measurements were made, and then the frequency for the board clock was updated via software command, and the effect on power measurement would be observed and recorded. What was not understood was that the Digital Locked Loop had to be reset in order to achieve clock synchronization within the FPGA design. Not achieving DLL lock-up results in excessive wait time on local address and data busses and wasted power consumption. The "hump" in power consumption at the FPGA below 60 MHz points out a condition where more power is wasted than really should be required by the programmed function.

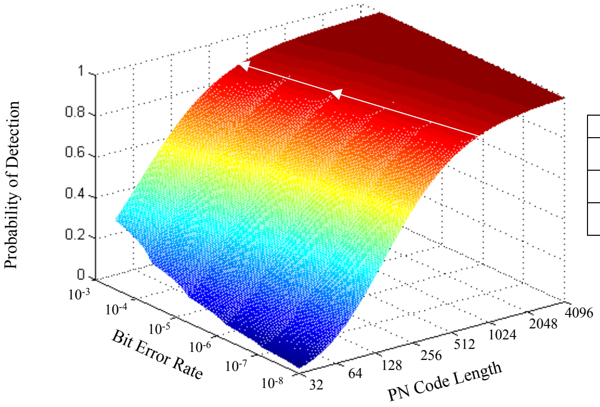
To resolve this problem and insure accurate data collection for this report, our measurement procedure was modified to re-programming the FPGA and board's clock generator for each change in frequency that was measured. Although this involved more steps than should be necessary, we were certain that the DLLs were locked up, and that the data presented in this research was valid. Future work must address ability of the board's interface VHDL or the software API to provide a mechanism to re-sync after a clock frequency change

# 9 Software Radio

## 9.1 Overview

A testbed system developed for smart networking radio algorithms is described in an IEEE paper written by AFRL researchers, John Patti, Bob Husnay and Joe Pintar[1]. Their modular software environment and Phase I hardware testbed provided a framework for the development of advanced processing algorithms, adaptive multirate systems, and operational radio algorithms and modules. The FLIPWAVE spread-spectrum modem invented at the U.S. Air Force Research Laboratory (AFRL) was developed and evaluated using that testbed. A unique feature of the Flip-Wave modem is a new single-channel quadraphase differential RAKE receiver processor

ISI was fortunate to work with the AFRL Rome Research Laboratory to obtain the software for the Flip-Wave modem and to examine that software and processing algorithm for use within PASPE. Our goal was to study the algorithm and produce multiple versions of the modem for varying quality versus power requirements. After study of the algorithm, our approach, as seen in Figure 59, was to adjust the bit-error-rate for a given probability of detection. Varying the pseudonoise (PN) code length from 1024 to 256 while holding probability of detection constant, results in a bit-error-rate that ranges from $10^{-8}$ to $10^{-3}$. The overlaid white line in Figure 59 shows that range. The predicted energy decrease associated with the decline in algorithm quality is due to the fact that processing required is relative to the PN code length.



| Pd | BER | PN | Energy |
|---|---|---|---|
| 0.90 | $10^{-8}$ | 1024 | $E_0$ |
| 0.90 | $10^{-5}$ | 512 | $0.8E_0$ |
| 0.90 | $10^{-3}$ | 256 | $0.6E_0$ |

**Figure 59 - Algorithm "Quality" Variability**

[1] A Smart Software Radio: Concept Development and Demonstration, John J. Patti, Robert M. Husnay, and Joseph Pintar. IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 17, NO. 4, APRIL 1999

Initial processing estimates, given the I & Q chip rate of 5 Million Chips per sec with 4 samples per PN code chip gave a 20 Million samples-per-second rate. The radio system requires a high dynamic range (12 bits min, 16 bits highly desirable). Using a variable processing gain with either 32, 128, 1024 PN chips per data bit leads to data rates of 312.5, 80, and 9.76 kbps. The data rate is calculated by data rate = chip rate/(chips/bit) * 2 bits/symbol. With a ½ rate Viterbi encoding, the data rates are 156.25, 40, and 4.38 kbps. For the ½ rate Viterbi, the data rate is calculated by data rate = chip rate/(chips/bit). This leads to the Adaptive Wideband Signal Processing Requirement on the order of 30 GOPS. Intelligent power management is essential.

The Communications Modem from the AFRL Flip-Wave Software Radio is comprised of two major functions: the Transmitter and the Receiver.

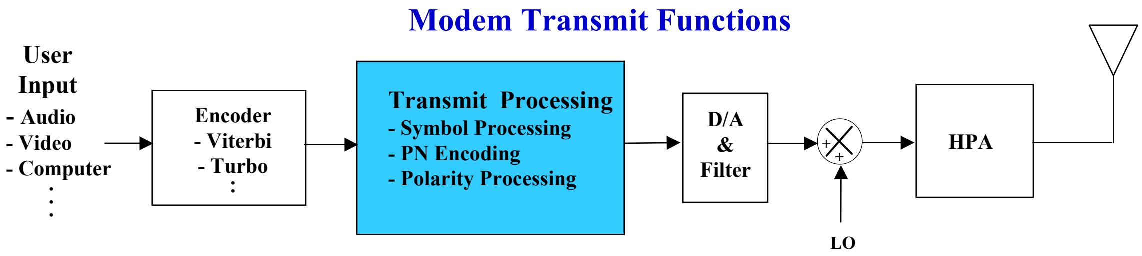


Figure 60 - Flip Wave Modem Transmit Functions

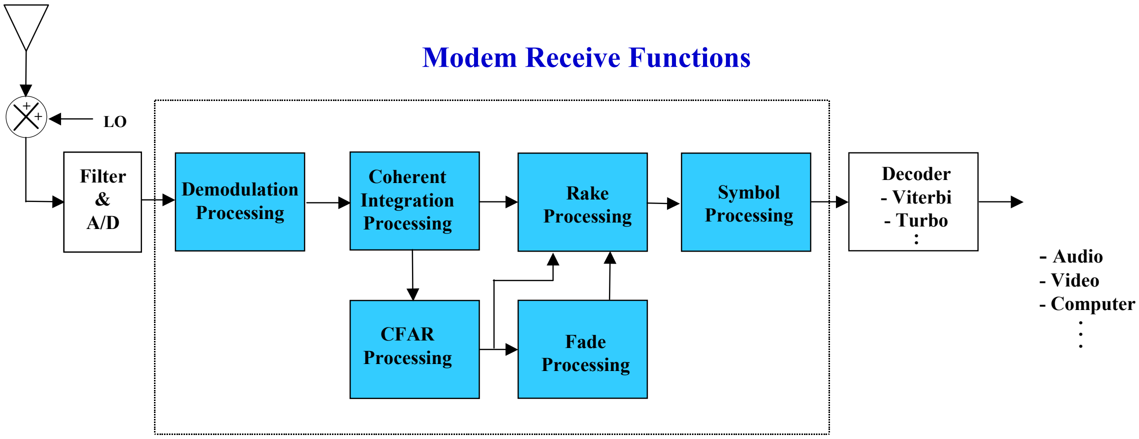


Figure 61 - Flip Wave Modem Receive Functions

Figure 60 shows the block diagram for the Transmit Functions of the Flip Wave Modem while Figure 61 shows the block diagram of the Receive Functions. The blue shaded sections correspond to the software code example received from AFRL.

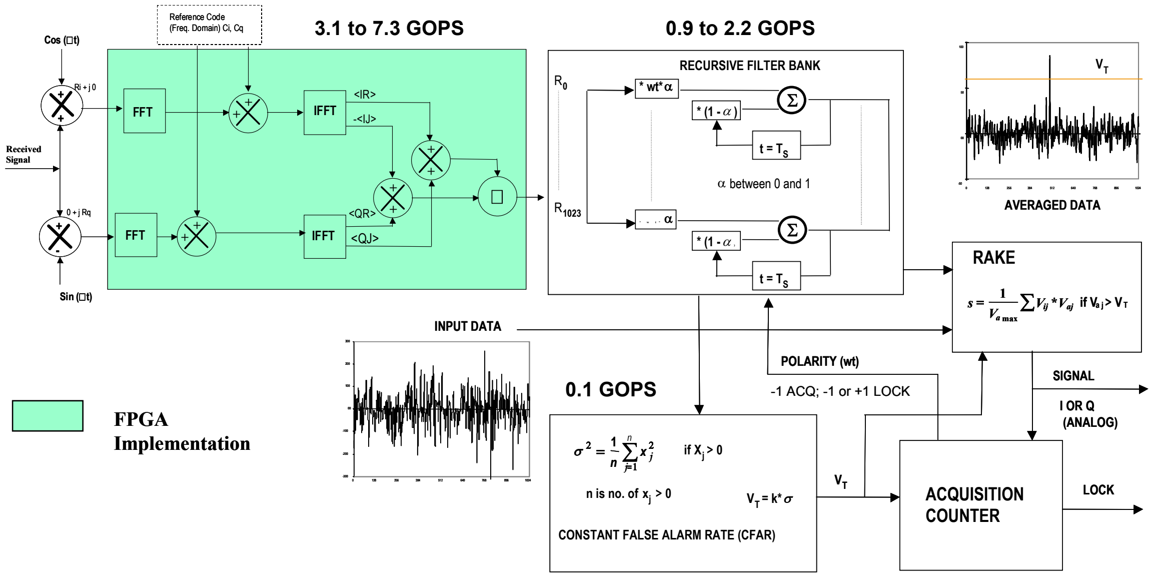


**Figure 62 - Flip-Wave Modem Processing Functions**

An expanded view of the blue shaded areas from the receive processing chain is shown in Figure 62. The Correlation Function was chosen for FPGA implementation due to that function including FFT processing, and since the nature of that processing is applicable to many other algorithms. The FPGA implementations is an FFT, a complex multiply, followed by an inverse FFT, and floating point complex data is used throughout. The Annapolis Microsystems radix-256 FFT core was to be used.

Although time did not permit, other FPGA implementations for the correlation function were considered, including a fixed point FFT implementation or other radix implementations for throughput improvements. Another alternative included a FIR Filter approach, which spread across the logic gates of the FPGA looked promising.

Other functions, such as the Recursive Filter Bank and Constant False Alarm Rate (CFAR) processing were examined, but time did not permit including implementing during this study.

## 9.2 Implementation for PASPE

The Receiver function became the Application and the steps described above were performed as follows:

- Determine what power consuming processing within the application lends itself to implementation in FPGAs.
- In the Receiver application, processing known as the “correlation function” lends itself to implementation in FPGAs. The correlation function has a typical signal processing flow: forward FFT -> complex multiply -> inverse FFT.
- Determine what parameter(s) in the application affect the power consuming processing.
- In the Receiver application, the size of the Pseudo Noise (PN) code is directly related to the size of the FFTs that are performed during the correlation function. The larger the PN Code size, the larger the FFTs. The larger the FFTs, the more computations are performed and the more power is consumed.
- Determine how many values/levels these parameters can have, how they interact with each other, which parameter is the "main" parameter, and how this parameter affects quality.

The Receiver application has 7 different PN code sizes. Therefore, the Receiver application will have 7 different quality versions. Within each PN code size, there are 7 different Desired Bit Error Rates (BER). Changing the Desired BER within the same PN code size does not affect power consumption, but does affect quality. The Receiver application manages the Desired BER within the PN code size.

PN16 - lowest power consumption, lowest quality version
PN32
PN64
PN128
PN256
PN512
PN1024 - highest power consumption, highest quality version

Desired BER $10^{-9}$ - lowest probability of overcoming fatal error vs. lowest bit error rate
Desired BER $10^{-8}$
Desired BER $10^{-7}$ - default
Desired BER $10^{-6}$
Desired BER $10^{-5}$
Desired BER $10^{-4}$
Desired BER $10^{-3}$ - highest probability of overcoming fatal error vs. highest bit error rate

The relationship between the PN code size and the Desired BER is as follows.

If a fatal error occurs during message processing, lowering the Desired BER within the PN code size increases the probability of receiving a message with no fatal errors, but also increases the number of bit errors in the message. Raising the Desired BER has the opposite affect. Fatal errors include:

- No acquisition lock
- No frame sync found
- Acquisition lock lost during message processing
- No End of Message (EOM) found
- Incomplete message

Determine what conditions would cause the application to need or want to request a change to its quality version. Also, determine what the scope of the change request will be.

The Receiver application uses the following conditions to determine the quality figure request output to the Application Manager and the value of the Desired BER:

<u>If a fatal error has occurred</u>

If the Desired BER is not $10^{-3}$
Output a Q_Delta request of "no change in PN code size" to the Application Manager
Change the Desired BER to the next lower value ($10^{-5}$ would change to $10^{-4}$)

<u>If the Desired BER is $10^{-3}$</u>

Output a Q_Delta request of "increase PN code size" to the Application Manager
If the Q_Delta request is granted, default the Desired BER to $10^{-7}$

<u>If a fatal error has not occurred</u>

If the Actual BER is unacceptable and the Desired BER is not $10^{-9}$
If a fatal error at next higher Desired BER has not occurred within the current PN code size more than once
Output a Q_Delta request of "no change in PN code size" to the Application Manager
Change the Desired BER to the next higher value ($10^{-7}$ would change to $10^{-8}$)

```
Else (a fatal error at next higher Desired BER has occurred within the current PN code size more than once)
    Output a Q_Delta request of “increase PN code size” to the Application Manager
    If the Q_Delta request is granted, default the Desired BER to $10^{-7}$

Else if the Actual BER is unacceptable and the Desired BER is $10^{-9}$
    Output a Q_Delta request of “increase PN code size” to the Application Manager
    If the Q_Delta request is granted, default the Desired BER to $10^{-7}$

Else (the Actual BER is acceptable)
    Output a Q_Delta request of “no change in PN code size” to the Application Manager
    Leave the Desired BER unchanged
```

There are no conditions in which the Receiver application would request a lower quality version, although the Application Manager may command the Receiver application to employ a lower quality version of itself. The conditions under which this may occur are defined in the Mission Parameters used by the Application Manager.

Determine what parameters and metrics the application will output/input to/from the Application Manager.

The Receiver application will output a quality figure change (Q_Delta) request. This request will indicate no change in current version or a request to change to the next highest version. The Receiver application will also output a small set of metrics (Fatal_Error, Desired_BER, and Actual_BER) that will be displayed on the Application Manager’s GUI for informational purposes. They may also be used in power event logging. The Receiver application will input a mode ID number from the Application Manager indicating what version of the application is to be run.

Determine what parameters need to be initialized/set up and what initialization processing needs to be performed if the version of the application changes.

In the Receiver application, the following parameters and processing are based on the PN code size and therefore, need to be initialized when a new version of the application is to be run:

- Free current buffers and re-allocate new for various vectors whose sizes are determined by PN code size
- Calculate receive weights
- Calculate reference codes and download to FPGA memory
- Perform Receiver stabilization processing once upon version change
- Load FPGA core with new version’s correlation function

Collect power and timing measurements for each version of the application as well as the retooling costs associated with changing from one version to another.

The Flip Wave modem was further split, using H-RTExpress, and run as four processing groups on the 4-node Linux cluster.

The Master process initializes and performs all socket communication with non-Receive functions. It performs receive start up initialization, and is responsible for performing the receive processing loop control (Seeker vs. Processor). It also performs application version recommendation and submode determination.

The Seeker process looks for acquisition lock and frame sync only.

The Message Processor processes the  remainder of A/D buffer looking for a message with end-of-message (EOM) characters.

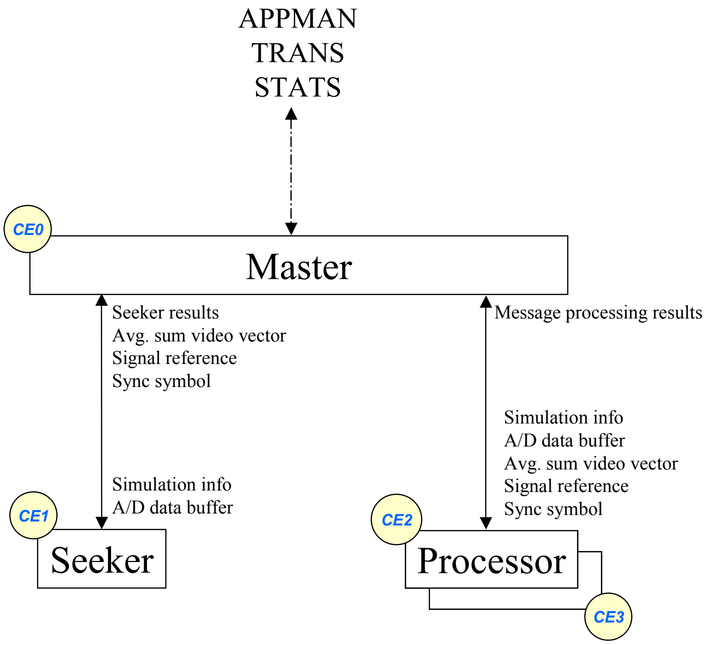


**Figure 63 - Flip-Wave Multi-Processor Implementation**

Figure 63, shows the receiver was broken in to four main pieces.  The Master process controlled the other processes and communicated to the outside world.  The Seeker process was responsible for scanning the input receive buffer to find the start of a message.  The Seeker performed the correlation function for the message preamble.  Once a valid start of message was found, the seeker would report the buffer address to the master, and the master was then responsible to assign a message processor, the third process, to decode the remainder of the message.  Since messages could overlap, if the seeker found another start of message within the buffer, the master could then assign the next processor in the list to decode that message.  The multi-process architecture would allow the modem to keep up with the incoming data stream and work on simultaneous messages.

Message processing also required the correlation function.  Since only one FPGA board was available, the simulation was run using the FPGA implementation of correlate with the seeker and then with the message processor separately.  The two message process functions used differed only in that one used the software implementation of correlate, while one used the FPGA implementation of correlate.

The processes appear in the right-hand window of the H-RTExpress “mapit” tool, as seen in Figure 64. There is one icon for each of the elements of the receiver, the Master, the Seeker, the Message Processor, and the Message Processor using an FPGA function.  The left-hand window of mapit shows the compute elements available.  In H-RTExpress the FPGA is thought of as a co-processor, or pre-processor, that is associated with a general-purpose processor, and the message passing interface (MPI) standard is used for communication between general-purpose processors.  The “enhanced” processor, meaning the compute element that is associated with the FPGA, is shown in Figure 64, and one of the message processing groups is mapped to that compute element.

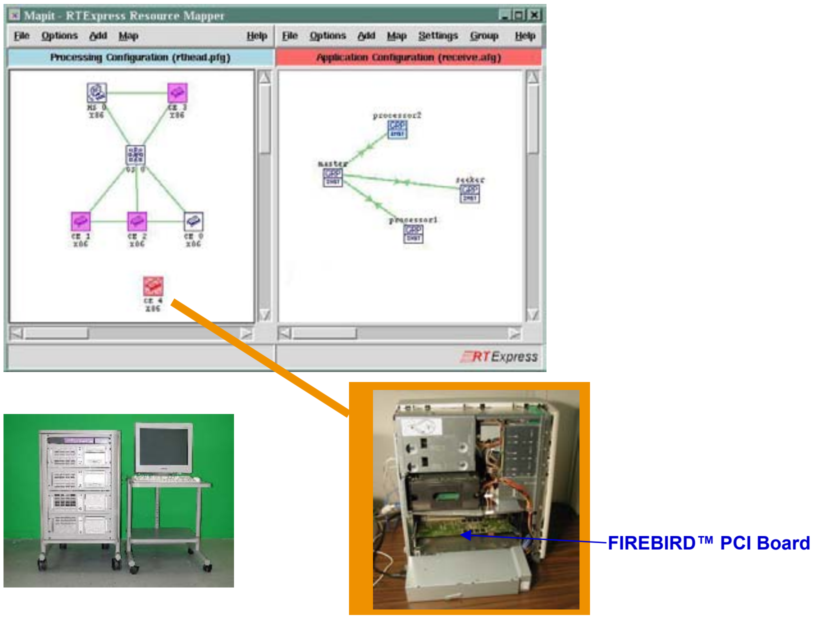
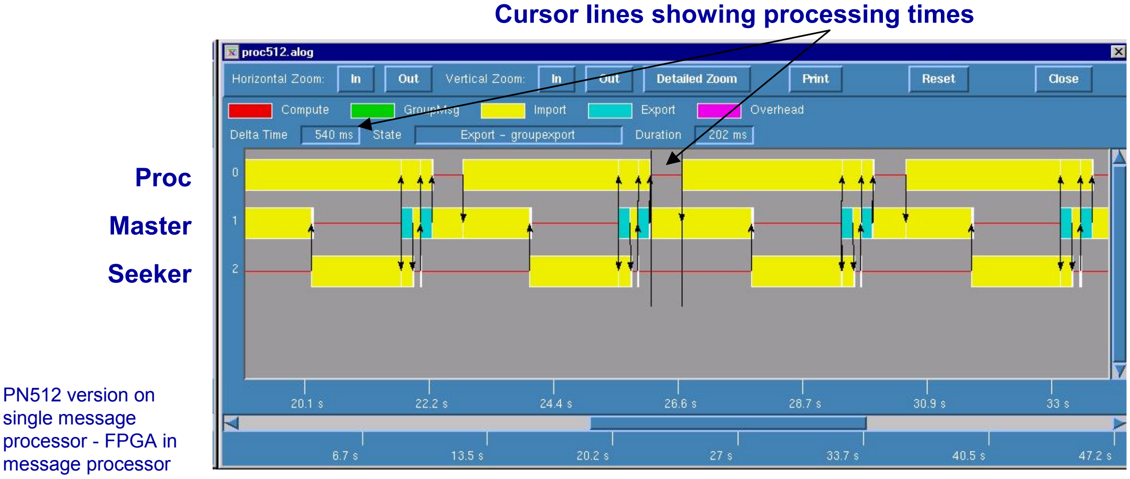

Mapit - RTExpress Resource Mapper
File Options Add Map Help
File Options Add Map Settings Group Help
Processing Configuration (rthead.pfg)
Application Configuration (receive.afg)
processor2
master
seeker
processor1
RTExpress
FIREBIRD™ PCI Board

**Figure 64 - H-RTExpress "mapping" to Linux cluster & FPGA board**

The features of H-RTExpress were relied on to load the correlation function into the FPGA, initialize the FPGA board, as well as close done the FPGA board on termination of the program.


proc512.alog
Horizontal Zoom: In Out Vertical Zoom: In Out Detailed Zoom Print Reset Close
🟥 Compute 🟩 GroupMsg 🟨 Import 🟦 Export 🟪 Overhead
Proc
Master
Seeker
0
1
2
20.1 s
22.2 s
24.4 s
26.6 s
28.7 s
30.9 s
33 s
6.7 s
13.5 s
20.2 s
27 s
33.7 s
40.5 s
47.2 s

Cursor lines showing processing times

Delta Time 540 ms State Export – groupexport Duration 202 ms

PN512 version on single message processor - FPGA in message processor

**Figure 65 - Post Mortem Dump (Flip Wave)**

Yellow region show waiting for data import between processors

Red bar is actual processing - AFRL Software Radio example software written in C and User C-files placed into MATLAB

### 9.3 Correlate function

Referring to Figure 62, the correlation function described in the receive processing chain was implemented in the FPGA. This function consisted of an FFT, frequency domain decimation, and an inverse FFT, as shown in Figure 66.

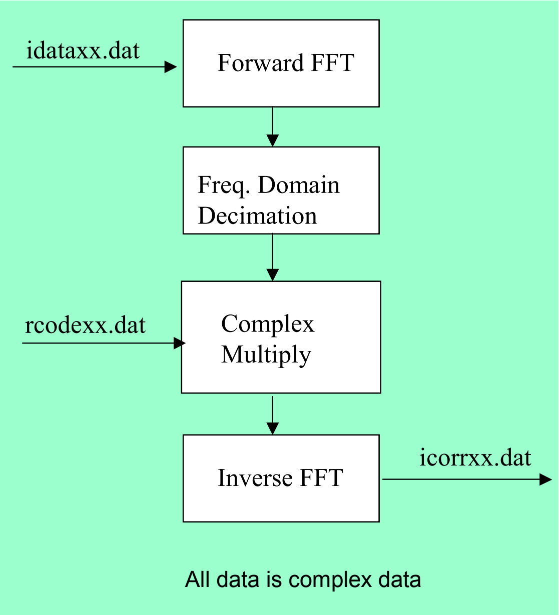


**Figure 66 - Correlate Function Block Diagram**

The correlate function was created for the FPGA using the Annapolis Microsystems COREFIRE tool. At the time this problem was being worked, COREFIRE was just being released to the general public.

The version of the FFT function has changed significantly since this release of COREFIRE. The version we used had a "token" interface, which used a four-bit number to identify each input and output spectrum. Each token could have its own set of parameters regarding the FFT to be performed. The FFT core was the complex-floating-point core described in the earlier section.

To produce "versions" of the correlate function that could trade the quality of processing against the power required, the FFT processing sizes were adjusted as described earlier through the use of various PN-code lengths. As seen in Figure 67, varying PN size from 16 to 1024 results in increasing FFT lengths. For example, the PN-256 (idata256) requires a 2048 point, complex floating point forward FFT and, due to the decimation process, a 1024 point, complex floating point inverse FFT.

| | Forward FFT | Inverse FFT |
|---|---|---|
| idata16 | 128 pt | 64 pt |
| idata32 | 256 pt | 128 pt |
| idata64 | 512 pt | 256 pt |
| idata128 | 1024 pt | 512 pt |
| idata256 | 2048 pt | 1024 pt |
| idata512 | 4096 pt | 2048 pt |
| idata1024 | 8192 pt | 4096 pt |

**Figure 67 - PN Code size vs. FFT Size**

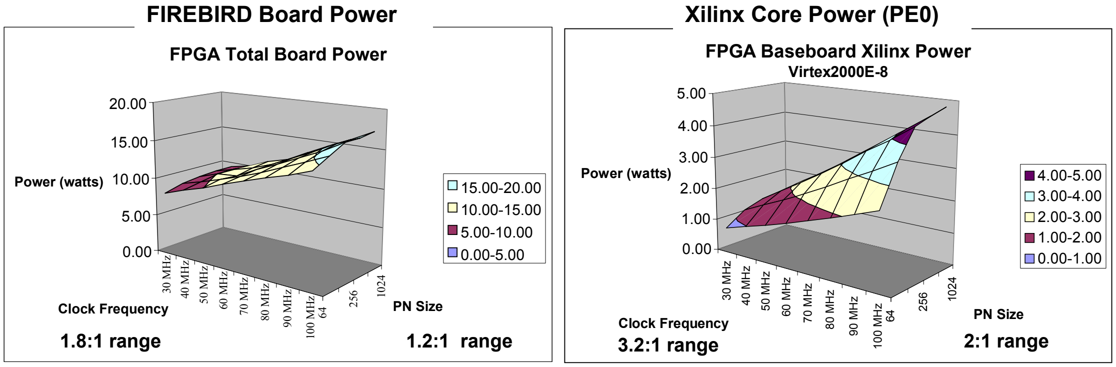
For our implementation, the Annapolis FFT core was radix-256, and therefore the smallest PN code length that could be processed was the PN-64. A C-code version of correlate was compared to the FPGA implementation. The results in Figure 68 show that for a PN-1024, the 500 MHz Pentium III took more than 18 msec to process, while then 100 MHz FPGA took 1 msec. As the size of FFT processing increased, the benefit of the FPGA implementation became more apparent. Rather than serial processing, the problem is spread across the gates of the hardware, allowing the slower clocked device to actually surpass the performance of the device clocked at a higher frequency.

The times in Figure 68 are in seconds and are times for one pass through the correlate function (1 channel). As shown in Figure 67, the forward FFT size is 8 times the PN Size and the inverse FFT size is 4 times the PN Size.

|  | GPP | FPGA | FPGA | FPGA |
|---|---|---|---|---|
| PN Size | 500 MHz | 30 MHz | 70 Mhz | 100 Mhz |
| 64 | 0.000842 | 0.000277 | 0.000214 | 0.000200 |
| 128 | 0.001831 | 0.000413 | 0.000270 | 0.000242 |
| 256 | 0.003954 | 0.000633 | 0.000407 | 0.000358 |
| 512 | 0.008611 | 0.001085 | 0.000667 | 0.000577 |
| 1024 | 0.018584 | 0.002000 | 0.001199 | 0.001029 |

**Figure 68 - Correlate Timing - FPGA vs GPP Comparison**

As in the FFT tests, the two “knobs” turned included clock frequency and PN-code size, which corresponded to FFT spectrum size. As frequency increased the power consumed increased, and Figure 69 shows the power consumption of the total FIREBIRD board and also of just the FPGA core for the baseboard Xilinx FPGA running the FFT. From the same figure, it is also seen that as PN Code size increased, so does power consumption.


FIREBIRD Board Power

Xilinx Core Power (PE0)

**Figure 69 - Correlate Function Power**

The measurement technique for the FIREBIRD board used actual voltage and current measurements as discussed earlier. Power on the General-Purpose Processor (GPP) was measured at the power supply as a difference in power consumed while at idle from the power consumed while executing the correlate function in an infinite loop. Power on the GPP was 16.5W at all sizes. Although it is difficult to compare the actual wattage numbers between the GPP and FPGA due to the difference in measurement techniques, it is important to note that there was no change in power due to PN Code size on the GPP. FPGA Power consumed increased with clock and with PN Code Size.

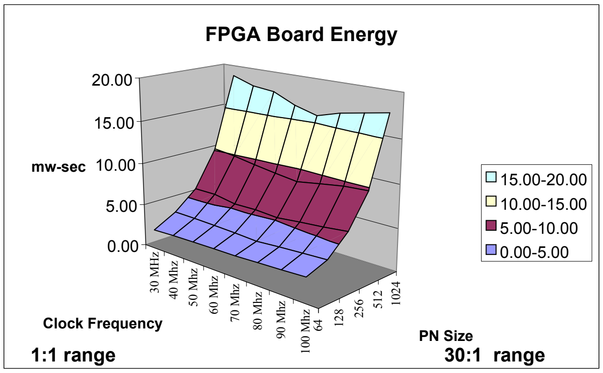
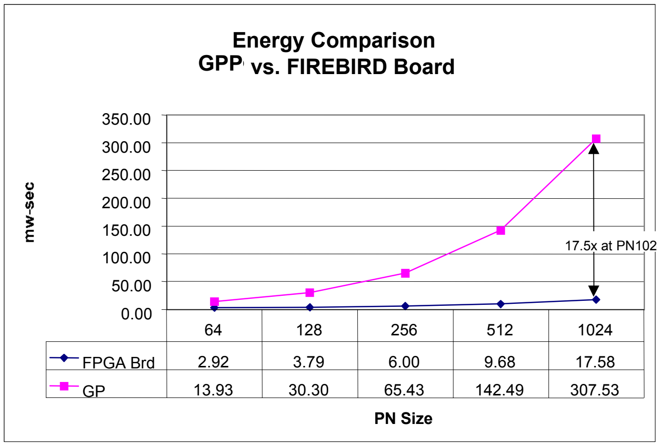


**Figure 70 - Correlate Function Energy**

Factoring in the time to process, Figure 70 shows the energy consumed by the correlate function. The curves are flattend in the frequency axis since the same amount of work is performed regardless of the speed to perform the work, however, a range of energy consumed is visible due to varying the PN Code size. The following figure, Figure 70, compares the GPP energy with the FPGA and the benefit of FPGA is seen at larger PN Code sizes where serial GPP operation time becomes long.



| | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|
| FPGA Brd | 2.92 | 3.79 | 6.00 | 9.68 | 17.58 |
| GP | 13.93 | 30.30 | 65.43 | 142.49 | 307.53 |

**Figure 71 - Energy Comparison of Correlate Function - GPP vs FPGA**

One remaining piece of information was needed for the Application Registry for the modem versions. The cost to re-tool is important to consider, especially if that cost erodes the gain expected from the version that is to be swapped in. To swap versions involved tasks on the GPP and the FPGA.

Re-tool time in GPP consists of:

- Weight generation
- PN Code generation
- Reference Code generation

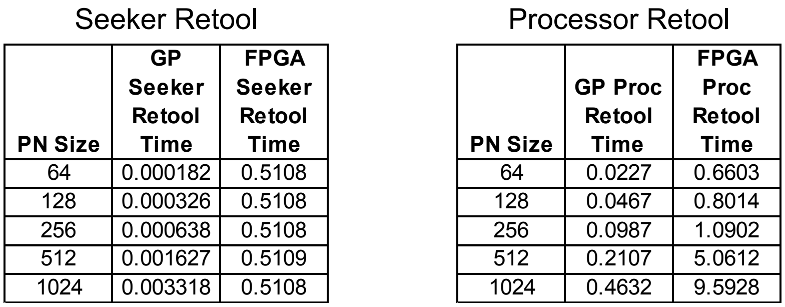
FPGA Retool Time consists of:

- FPGA Retooling Time and Power includes:
- Deprogram Baseboard PE
- Deprogram IO PE
- Program Baseboard PE (init.x86, core.x86)
- Program IO PE (fbcorrlateS.x86)
- Reset IO PE

- Configure Forward FFT
- Configure Inverse FFT
- Enable IO PE
- Download Rcode, or reference codes,  to memory (1 symbol worth)
- Determine counts
- Write counts to registers
- Write to token registers

FPGA Measured Power for retool is same as “safe core” power measurement.

The correlate function was used both in the Seeker function and the Message Processor function. The times shown in Figure 72 are in seconds.  The timing was with the FPGA clocked at 100 MHz.   The majority of the time is taken up transferring data to the FPGA board.  The reference codes are loaded into local memory on the FPGA board for use in the complex multiply and the amount of data to store is proportional to the PN Code size selected.



Seeker Retool

| PN Size | GP Seeker Retool Time | FPGA Seeker Retool Time |
|---|---|---|
| 64 | 0.000182 | 0.5108 |
| 128 | 0.000326 | 0.5108 |
| 256 | 0.000638 | 0.5108 |
| 512 | 0.001627 | 0.5109 |
| 1024 | 0.003318 | 0.5108 |

Processor Retool

| PN Size | GP Proc Retool Time | FPGA Proc Retool Time |
|---|---|---|
| 64 | 0.0227 | 0.6603 |
| 128 | 0.0467 | 0.8014 |
| 256 | 0.0987 | 1.0902 |
| 512 | 0.2107 | 5.0612 |
| 1024 | 0.4632 | 9.5928 |

**Figure 72 - Correlate Function - Retool Times**

## 10 Software Radio Demonstrations

The software radio demonstrations exercised the Flip-Wave modem as described in the section 9.  The PAGUI tool managed the scenario control, providing stimulus and noise levels as needed, along with controlling power supply or battery conditions using the SIMBAT tool.

Primarily, the demonstrations focused on the Application Manager’s ability to react to the environment as known through the power condition monitoring, application performance metrics, and known states, or application versions, and make decisions to swap state to another application version.  Several Mission Profile rule-sets were created to illustrate and test this ability.

### 10.1 Mission Profile Descriptions

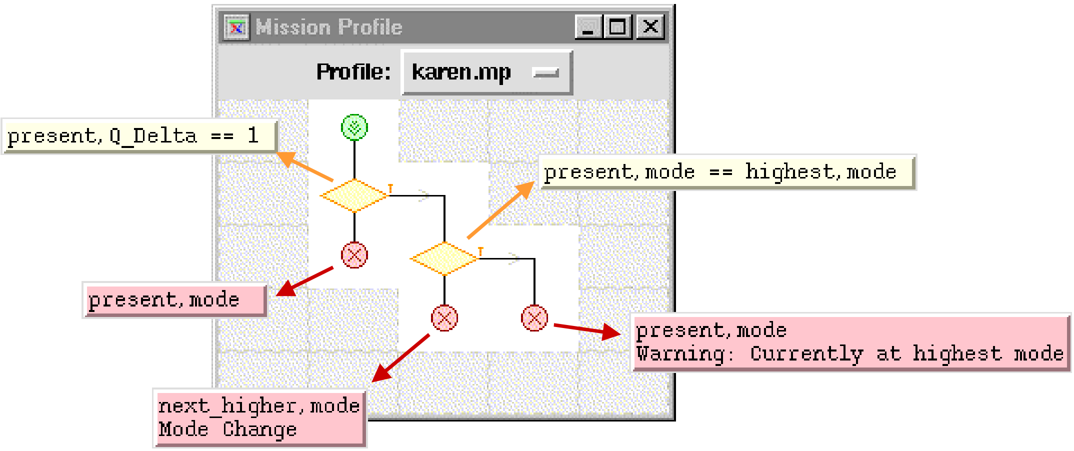


**Figure 73 - Mode Request Change regardless of Power and/or Time**

In the mission profile shown in Figure 73, the Application Manager grants the mode request change received from the Application regardless of power and/or time.

Refer to **Appendix 4** for more information on the Software Radio Modes and Submodes.

The Application Manager's processing flow is as follows:

```
If the Quality Delta Request received from the Application indicates no change
    The Application Manager will not change the mode

Else if the Quality Delta Request received from the Application indicates a
change
    If the present mode is at the highest mode possible
        The Application Manager will not change the mode and "Warning: Currently
        at highest mode" will be displayed on the Simulation Control GUI
    Else
        The Application Manager will change the mode to the next higher mode
```

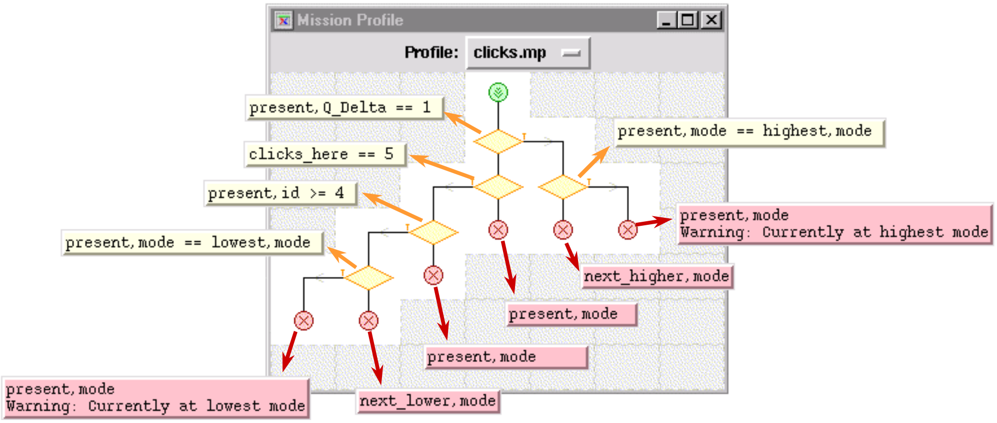


**Figure 74 - Mode Change based on time and Application Request**

In the mission profile shown in Figure 74, the Application Manager determines mode change based on time as well as the mode request received from the Application.

The Application Manager's processing flow is as follows:

```
If the Quality Delta Request received from the Application indicates change
   If the present mode is at the highest mode possible
      The Application Manager will not change the mode and “Warning: Currently
      at highest mode” will be displayed on the Simulation Control GUI
   Else
      The Application Manager will change the mode to the next higher mode
Else
   If the number of clicks in present mode is 5
      If the present mode is greater the PN128
         The Application Manager will not change the mode
      Else
         If the present mode is at the lowest mode possible
            The Application Manager will not change the mode and “Warning:
            Currently at lowest mode” will be displayed on the Simulation
            Control GUI
         Else
            The Application Manager will change the mode to the next lower
            mode
   Else
      The Application Manager will not change the mode
```

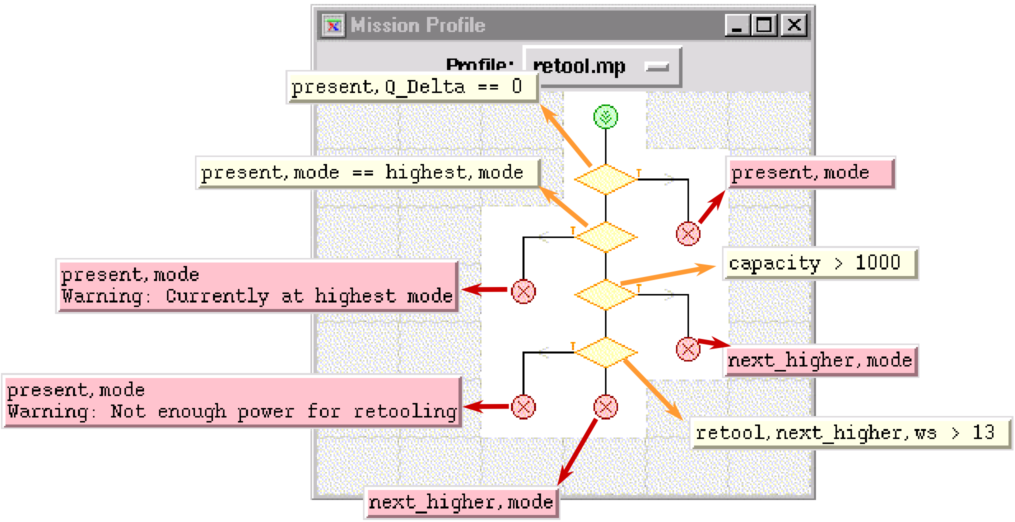
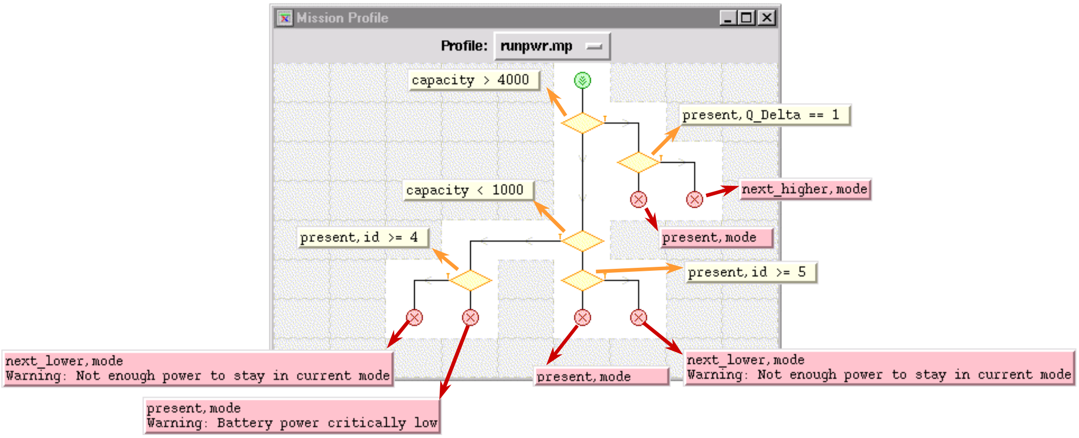


**Figure 75 - Mode Change based on Retooling power required and Application Request**

In the mission profile shown in Figure 75, the Application Manager determines mode change based on the desired mode's re-tooling power required as well as the mode request received from the Application.

The Application Manager's processing flow is as follows:

```
If the Quality Delta Request received from the Application indicates no change
    The Application Manager will not change the mode
Else
    If the present mode is at the highest mode possible
        The Application Manager will not change the mode and "Warning: Currently
        at highest mode" will be displayed on the Simulation Control GUI
    Else
        If the current battery capacity is greater than 1000 ws
            The Application Manager will change the mode to the next higher mode
        Else
            If the power required to retool to the next higher mode is greater
            than 13 ws
                The Application Manager will not change the mode and "Warning: Not
                enough power for retooling" will be displayed on the Simulation
                Control GUI
            Else
                The Application Manager will change the mode to the next higher
                mode
```



Profile: runpwr.mp

**Figure 76 - Mode change based on present mode's runtime power Required**

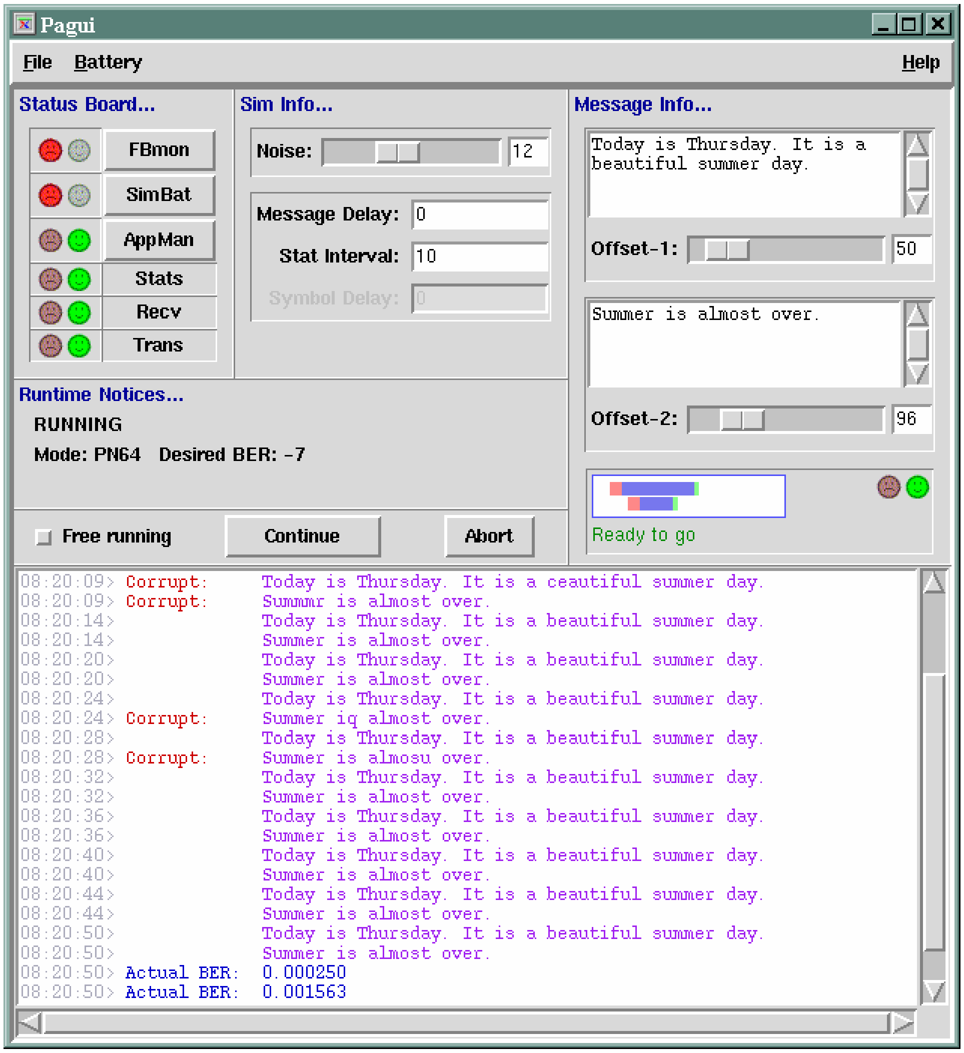
In the mission profile shown in Figure 76, the Application Manager determines mode change based on the present mode's run-time power requirement as well as the mode request received from the Application.

The Application Manager's processing flow is as follows:

```
If the current battery capacity is greater than 4000 ws
    If the Quality Delta Request received from the Application indicates change
        The Application Manager will change the mode to the next higher mode
    Else
        The Application Manager will not change the mode
Else
    If the current battery capacity is less than 1000 ws
        If the present mode is PN128, PN256, PN512, or PN1024
            The Application Manager will change the mode to the next lower mode
            and "Warning: Not enough power to stay in current mode" will be
            displayed on the Simulation Control GUI
        Else
            The Application Manager will not change the mode and "Warning:
            Battery power critically low" will be displayed on the Simulation
            Control GUI
    Else
        If the present mode is PN256, PN512, or PN1024
            The Application Manager will change the mode to the next lower mode
            and "Warning: Not enough power to stay in current mode" will be
            displayed on the Simulation Control GUI
        Else
            The Application Manager will not change the mode
```

## 10.2 Scenario control / PAGUI

The Software Communications Modem stimulus scenarios are set up and controlled using the Simulation Control GUI (PAGUI) shown in Figure 77.



Noise: 12

Message Delay: 0

Stat Interval: 10

Symbol Delay: 0

Today is Thursday. It is a beautiful summer day.

Offset-1: 50

Summer is almost over.

Offset-2: 96

Ready to go

RUNNING

Mode: PN64   Desired BER: -7

```
08:20:09> Corrupt:     Today is Thursday. It is a ceautiful summer day.
08:20:09> Corrupt:     Summmr is almost over.
08:20:14>              Today is Thursday. It is a beautiful summer day.
08:20:14>              Summer is almost over.
08:20:20>              Today is Thursday. It is a beautiful summer day.
08:20:20>              Summer is almost over.
08:20:24>              Today is Thursday. It is a beautiful summer day.
08:20:24> Corrupt:     Summer iq almost over.
08:20:28>              Today is Thursday. It is a beautiful summer day.
08:20:28> Corrupt:     Summer is almosu over.
08:20:32>              Today is Thursday. It is a beautiful summer day.
08:20:32>              Summer is almost over.
08:20:36>              Today is Thursday. It is a beautiful summer day.
08:20:36>              Summer is almost over.
08:20:40>              Today is Thursday. It is a beautiful summer day.
08:20:40>              Summer is almost over.
08:20:44>              Today is Thursday. It is a beautiful summer day.
08:20:44>              Summer is almost over.
08:20:50>              Today is Thursday. It is a beautiful summer day.
08:20:50>              Summer is almost over.
08:20:50> Actual BER:  0.000250
08:20:50> Actual BER:  0.001563
```

**Figure 77 - Power Aware Simulation Control GUI (PAGUI)**

The Status Board section

- provides the ability to launch various functions needed by the radio: FBMON, SIMBAT, APPMAN, Statistics, Transmitter, and Receiver.
- provides status of each function is provided (happy green face for launched and sad red face for not launched).

The Sim Info section

- provides the ability to set the noise level injected into the transmitted message buffer (in –dB).
- provides the ability to set the statistics interval. The statistics interval is how often (every X messages) the bit error rate and other metrics are provided to the Application Manager.
- provides the ability to set the message delay and symbol delay. Note: these are not currently implemented.

The Message Info section

- provides the ability to input 2 messages for transmission/receipt.
- provides the ability to set the starting symbol offset for each message.
- provides status as to whether the messages and their respective offsets allow for valid transmission (ie: messages fit in buffer, message preambles don't overlap, etc).
- see Figure 78 for more details on message format

The Runtime Notices section

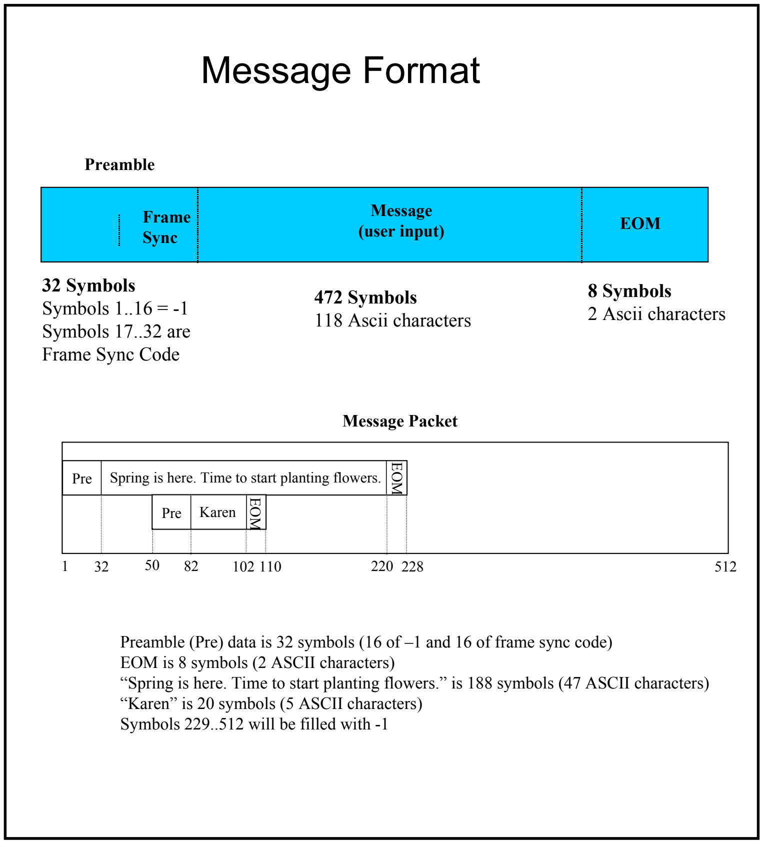
- provides run time status and notices such as the current Mode and Desired Bit Error Rate (BER), warning messages, etc

There are 3 buttons to control runtime operation

- Free running (toggle) – If on, the transmitter builds and transmits messages continuously.
- Continue – If Free running is not selected, clicking on Continue will cause the transmitter to build and transmit one message buffer.
- Abort – aborts radio operation.

Message status section (scrolling window)

- provides runtime status of receive message processing. Each message that is received and processed is displayed along with an indication as to the success of the receipt (message was corrupt, a fatal error occurred during processing, etc).
- provides Actual Bit Error Rate (BER) at each statistics interval for the messages processed in the interval.



## Message Format

Preamble

Frame Sync

Message
(user input)

EOM

**32 Symbols**
Symbols 1..16 = -1
Symbols 17..32 are
Frame Sync Code

**472 Symbols**
118 Ascii characters

**8 Symbols**
2 Ascii characters

**Message Packet**

Pre | Spring is here. Time to start planting flowers. | EOM

Pre | Karen | EOM

1 32 50 82 102 110 220 228 512

Preamble (Pre) data is 32 symbols (16 of –1 and 16 of frame sync code)
EOM is 8 symbols (2 ASCII characters)
“Spring is here. Time to start planting flowers.” is 188 symbols (47 ASCII characters)
“Karen” is 20 symbols (5 ASCII characters)
Symbols 229..512 will be filled with -1

**Figure 78 - Software Radio Message Format**

Figure 79 depicts the relationship between the PASPE Components (FBMON, SIMBAT, APPMAN, App Registry, Mission Profile, Data Log), the Application (Receive function of the Software Radio), PAGUI, the SW Radio Transmitter function, and the SW Radio Statistics Function. They communicate with each other via tcp/ip sockets.

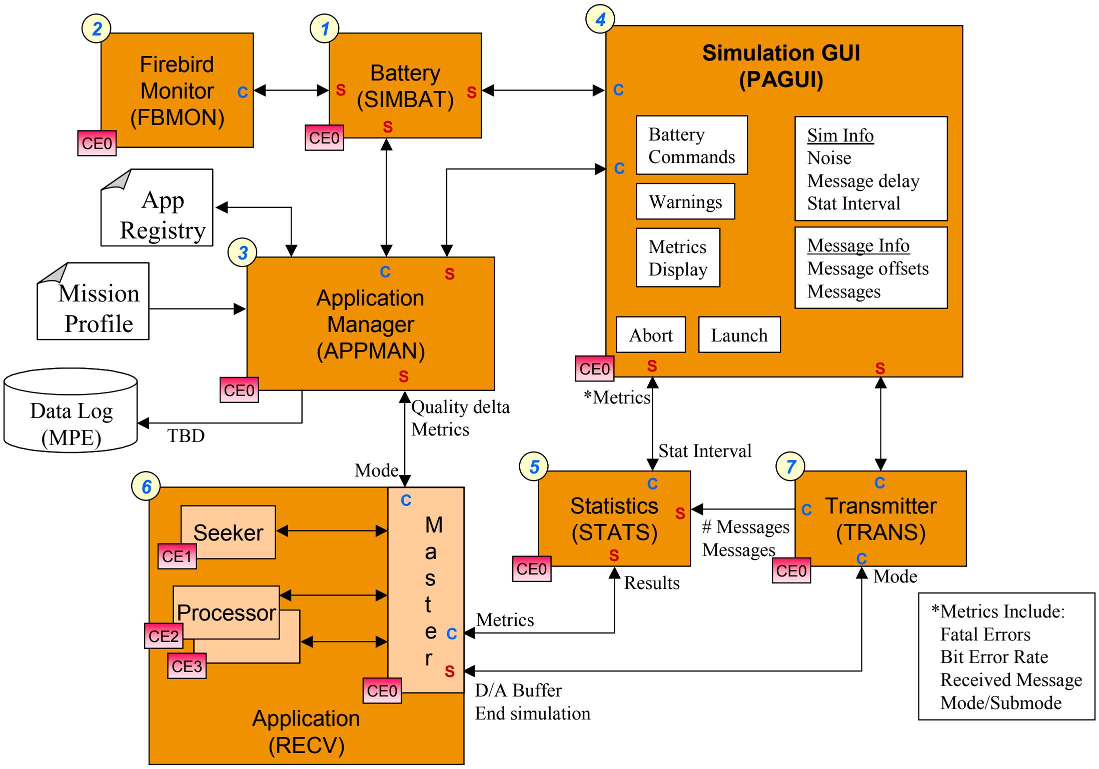


**Figure 79 - Simulation Setup**

Figure 80 is actually the same as the prior figure, however, the actual tool user interface is shown to better bring together the relationship of the PASPE tools discussed in this report as they were utilized for demonstrations and data gathering.

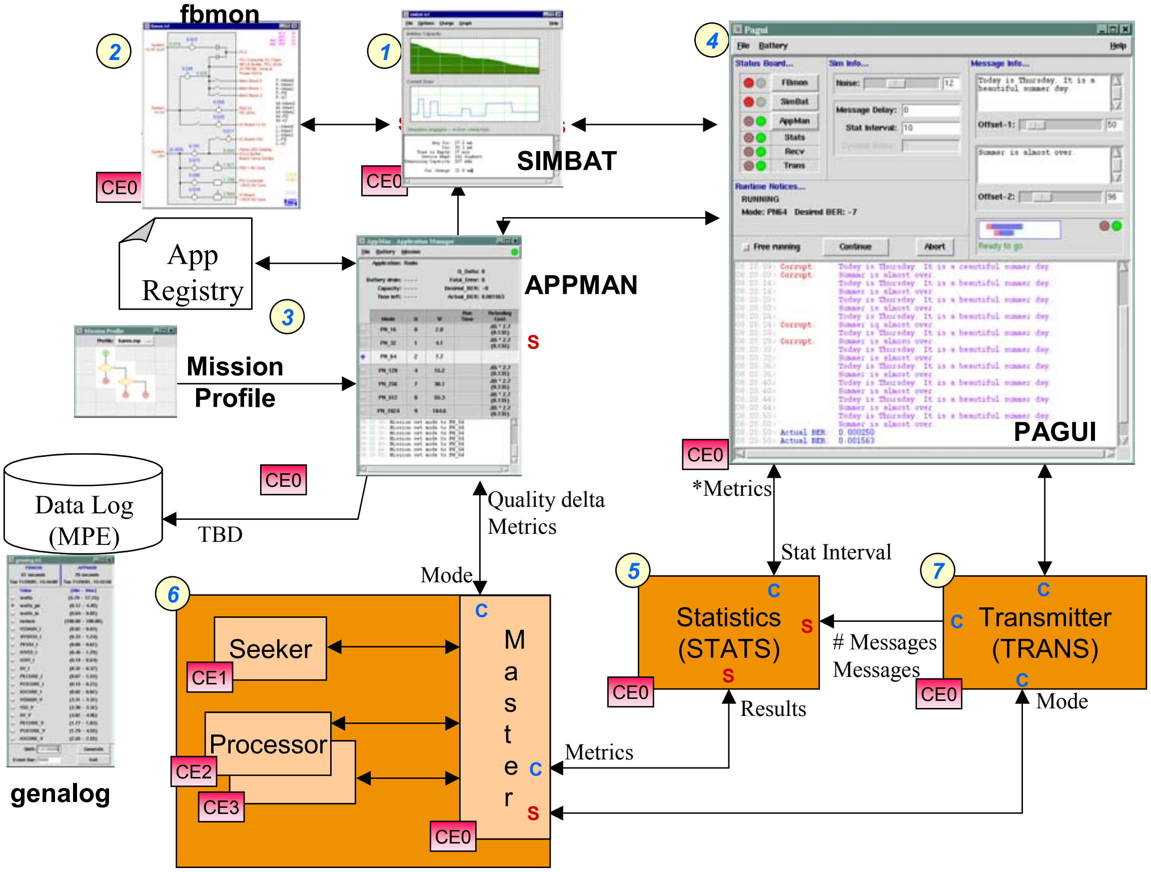


**Figure 80 - Simulation Setup (showing tools)**

# 11 Summary

## 11.1 Accomplishments

The Power Aware Signal Processing Environment involved the following Hardware and Software developments activities.

Adaptive Computing Systems (ACS) Hardware for DoD applications with Power Aware functionality was built by Annapolis Microsystems. The new FPGA board design was modified to include a software controlled power grid and programmable clocks, and the additional functionality is retained in the commercial offering of the Firebird board.

A software application-programming environment was created using a high-level language development system. (MATLAB, H-RTExpress). Software middleware (API) to communicate with the ACS hardware was also developed. A run-time system was developed, consisting primarily of Application Manager and Power Monitor. The environment also includes power profiling tools for evaluating power use.

The software environment was designed to support a heterogeneous mix of Power Aware devices, including Linux Clusters with ACS hardware, and embedded systems with ACS hardware. A logical extension for PASPE would include supporting System on a Chip (SoC) configurations.

To test the PASPE idea on a real-world problem, a communications modem application was planned as a demonstration vehicle. The software came from AFRL Rome Research Site and was adapted for use in this project. A Communications Modem Application was selected, and through collaboration with the Air Force Research Laboratory (AFRL), Information Directorate at the Rome Research Site, their Flip-Wave Spread Spectrum Modem (J. Patti, R. Husnay) example was used and modified with PASPE to achieve power demand adaptively and environmental adaptively.

In addition, a simulation capability was built to control the power environment and the radio and communications system environment. An Application Simulator was created to provide stimulus to the radio application with the capability to produce one or more messages with noise and interference. The message content is variable, and messages may overlap, be simultaneous, or be separated by time.

In addition, a Power Environment Simulation was created to simulate a battery-powered environment and to communicate to the run time system with "Smart Battery" messages.

The team completed the software development of the PASPE tools, and received the FIREBIRD board from Annapolis Microsystems. The hardware enhancements for Power Grid control and power measurements were verified through the use of the Annapolis software API and VHDL models.

Several FPGA cores were developed for testing, and core development was completed with Annapolis Microsystems utilizing the Annapolis Microsystems COREFIRE™ data flow FPGA Programming Tool. Rather than using the more lengthy VHDL development process, COREFIRE facilitated quick development of several FPGA images that were used for power and performance profile data collection measurements. Some of the cores developed and measured included simple add and multiply functions, FFT, and a correlation function.

The demo application using the AFRL Flip-Wave technique was completed along with the simulation stimulus and control software. The application was hosted in a parallel processing environment to enhance the ability to process multiple simultaneous messages, and to allow the opportunity for resource management of multiple processing nodes.

The Flip-Wave modem application was analyzed for Quality of Service Controls and subsequent Power Profiles. Several application "versions" were created and the PASPE Application Registry was populated with information on theses available versions.

The PASPE software tools were delivered with H-RTExpress to AFRL Rome Research Site and installed on the AFRL Linux cluster.

Power saving gains were achieved by adapting the application algorithm through the use of the Application Manager, and additional gains through the management of resources had also been projected. Research using FIREBIRD power grid to manage on-board resources produced minor gain, since the application, as implemented, required all components of the board to operate. Further, the board design was well thought out to conserve power. For instance, to remove power from unused memories might save power, however, the board logic already was removing the memory chip-select when the memory was not in use, thereby preventing logic gate toggles and power consumption. Measuring power with and without the memories enabled demonstrated little change in power consumption. Reloading the memory after a power-on would be costly and not justify the minor savings by shutting down the memory.

Another proposed idea was to disable one FPGA that processed a message while the other FPGA on board would search for a message present. In this application, the correlation function logic was in one FPGA while the FFT required by the correlation function was in the other FPGA. The correlate function is always in use whether searching for or processing a message, and therefore, no advantage was able to be realized by managing at the component level.

The receiver application was placed into a parallel processing environment with a master process, a message seeking process, and a message decoding process. Assuming that message decoders could be turned on only when required for processing an incoming message, time-line analysis from the software radio example shows varying power consumption results due to environment, message length, time between messages. This analysis is too dependent on selected time intervals between messages, and additional research is required to understand typical traffic scenarios. The power measured using a "safe core" was 5Watts. The "safe-core" power may be averaged with board processing powers ranging from 8 to 17 W, depending on application version (as driven by environment). The average may be weighted by time-off applying to the safe-core value, and time-on applying to the message processing power to arrive at an overall power for a given message scenario. Without managing resources, the overall power would be the message processing power. The impact, or savings, due to managing resources is then totally dependent on the selected scenario.

Merging of MPI Performance Evaluation (MPE) Post Mortem Dump log data with Power Events from power logging functions was completed to allow rendering of power events for evaluation using the "rteshot" tool. This graphic tool plots processing functions in a parallel system along a time scale, and the power usage monitored is placed graphically on the same time scale. This presentation has the benefit of observing the portions of processing that drive power usage.

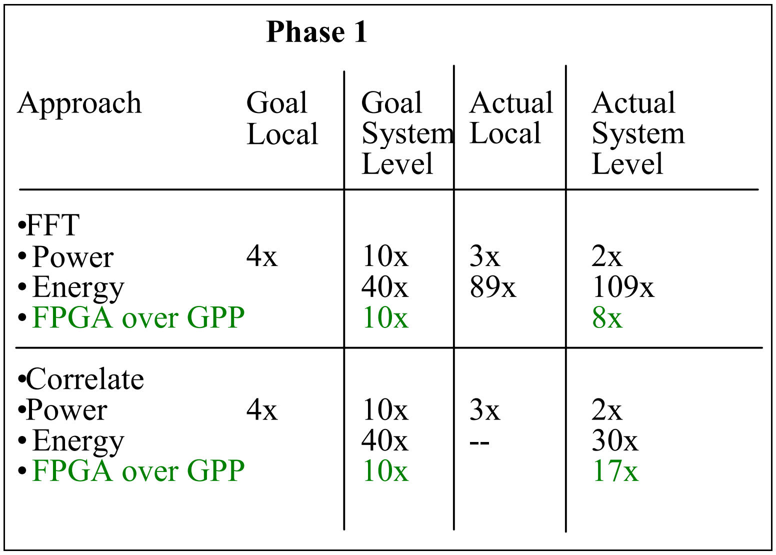
To specify a power aware software middleware for the SLAAC API, function prototypes were sent to USC/ISI for their consideration based on work with the FIREBIRD board and PASPE system. USC/ISI is working toward a standard Power Aware API.

In the course of this investigation several challenges were encountered. Annapolis Micro Systems provided excellent support for their new software and new hardware, however, complications are present with any new development. The FPGA 32-bit, floating-point, complex, FFT implementation required for this study had to be rebuilt for the new ACS hardware. The FIREBIRD FFT core was based on the WILDSTAR PCI radix-32 and radix-256 cores, however, the FIREBIRD PCI hardware utilized different communication methods between the baseboard Xilinx part for the FFT and the I/O board Xilinx part. The first iteration of the correlation function was completed at Annapolis on a WILDSTAR board, and therefore rework and additional learning was required to move the early WILDSTAR correlate function to the FIREBIRD. The FFT interface was completely different between WILDSTAR and FIREBIRD (client interface versus token interface) which, in turn drove redesign of the correlate function.

The first FIREBIRD FFT core was not operational to the full clock speed of the board, and the present core high end is 110 MHz. rather than 150 MHz. AMS also specified a minimum operating frequency of 60MHz due to low end of the DLL specification. In tests, the FFT results were accurate from 30 MHz through 110 MHz.

Recollection of data at some points became necessary due to requirement for DLL lock-up not being met at the time first measurements were taken

The chart shown in Figure 81 is patterned after the standard reporting format that was requested of all PAC/C projects. The chart was difficult to populate since this was not a project, such as a new processor or a new compiler, that would allow running with and without power aware features. The closest comparison to that is to compare the FPGA performance to the General Purpose Processor. We have done that, however, as noted earlier in this report, the comparison should be to observe the trend in power consumption rather than the absolute numbers due to differences in power measurement technique on each platform. The range of power and energy consumption based on turning knobs relative to the quality of performance of the application are highlighted in this investigation. The two major areas, an FFT Function, and a Correlation Function, are shown. The main knob adjusted was spectrum length for the FFT and PN Code length for the correlate function.



**Phase 1**

| Approach | Goal Local | Goal System Level | Actual Local | Actual System Level |
|---|---|---|---|---|
| •FFT | | | | |
| • Power | 4x | 10x | 3x | 2x |
| • Energy | | 40x | 89x | 109x |
| • FPGA over GPP | | 10x | | 8x |
| •Correlate | | | | |
| •Power | 4x | 10x | 3x | 2x |
| • Energy | | 40x | -- | 30x |
| • FPGA over GPP | | 10x | | 17x |

**Figure 81 - Phase 1 Status**

Measurement of a 32-bit floating-point complex FFT core demonstrated a 89x reduction in energy at the FPGA and 109x reduction at the FPGA board level as spectrum length was varied from 32K-point to 256-poing FFT. A loose comparison to the General Purpose Processor provided a 8x energy improvement using the FPGA board over the GPP for the 32K FFT, mainly due to the difference in processing time required on the FPGA.

Similarly, the correlate function (FFT, decimation, complex-multiply, inverse-FFT) demonstrated a 30x range of energy reduction as the Flip-Wave PN-code length was varied from PN-64 to PN-1024. Running the same function on the GPP, at 17x improvement in energy resulted.

## 11.2 Future Efforts

A FIR Filter FPGA Implementation of Correlate Function should be implemented to compare with the FFT based implementation. The time domain solution may have additional power savings by exploiting parallelism of FPGA logic for time performance, as compared to a serial operation in a general purpose processor. Other areas in the Flip-Wave processing, including the Recursive Filter Bank and the Constant False Alarm Rate (CFAR) processing sections, were identified as candidates for FPGA implementation and profiling, but time did not permit it.

The Application Manager was demonstrated to successfully swap versions for power advantage, however, further optimization of re-tool functions is required.

In the parallel processing environment used in this testing, only one general-purpose processor was outfitted with the FPGA board. Multiple message processors and seeker processors need to have multiple "enhanced" processors for parallel operation of the FPGA functions within the cluster.

The PASPE Application Manager and Mission Profile work together to manage the power consumed by an application. The Mission Profile contains information on the available "states" or versions of the application available. Mission Profiles are stored as text files and may be changed through a menu command. Another PAC/C research group from the University of California at Irvine has created the IMPACCT Tool for Mission Analysis. Rather than create Mission Profiles using the PASPE editor, the IMPACCT Tool may have potential to create the Mission Profiles rule sets for use by PASPE, or direct the selection of pre-stored rule sets based on the Mission Analysis time line information from IMPACCT.

Another potential tool that may be integrated with PASPE is the EDAptive Computing ASSISTANT (Embedded Micro Architecture SWEPT Simulator and Integrated Design Space Exploration and Tradeoff Analysis Tool).

The FPGA manufacturers are continuously developing parts with faster clock rates and additional gates and complexities. Some of the newer designs contain multipliers in addition to the look up table logic. A comparison of power and performance profiles using newer Xilinx Virtex-II FPGA (Wildstar-II expected 1Q02) to the data recorded in this effort should be accomplished to understand the direction the new parts are taking.

Annapolis Microsystems is furthering the development of COREFIRE and their library elements. A COREFIRE Fixed-Point FFT (FFT core module due 1Q02) should be compared to the data taken here on the 32-bit, floating point FFT core.

In addition to COREFIRE, other tools, including standard VHDL development, exist for creating FPGA images. The COREFIRE power and performance should be compared to that obtained when using AccelChip's AccelFPGA which takes MATLAB source to VHDL (based on Northwestern University MATCH Compiler). A similar comparison may be made using the Xilinx System Generator for Simulink for FPGA development.

The ISI team participated in the Smart Weapon/UAV IAT, and was exposed to the MATLAB and C source from that group. Time did not permit our team to full participate with an FPGA example. Determination of the "knobs" of operation (investigate ability affect spectrum size, for instance) to affect power consumption must be completed to understand how to apply PASPE to this problem.

The PASPE tools may be applied to other FPGA hardware in addition to the FIREBIRD. Other candidates include the Annapolis WILDSTAR and the USC/ISI SLAAC hardware.

# 12 Appendix 1 - PASPE Application Template

Putting an Application into the Power Aware Signal Processing Environment

1. Determine what power consuming processing within the application lends itself to implementation in FPGAs.
2. Determine what parameter(s) in the application affect the power consuming processing.
3. Determine how many values/levels these parameters can have, how they interact with each other, which parameter is the "main" parameter, and how this parameter affects quality. The outcome of this exercise determines how many quality versions of the application are needed. This information will be used in the Application Registry.
4. Determine what conditions would cause the application to need or want to request a change to its quality version. Also, determine what the scope of the change request will be (i.e. next higher or lower version, or specific version, etc.).
5. Determine what parameters and metrics the application will output/input to/from the Application Manager. Outputs will be used in the Application Registry and will be displayed on the Application Manager's GUI.
6. Determine what needs to be initialized/set up if the version of the application changes (FFT sizes, buffer sizes, noise tables, weight tables, etc.). The outcome of this exercise will determine both the "initialization" and "shutdown" portions (retooling) of the application.
7. Collect power and timing measurements for each version of the application as well as the retooling costs associated with changing from one version to another. This information will be used in the Application Registry.

# 13 Appendix 2 – SLAAC API

## 13.1 Power Monitor Functions

### 13.1.1 ACS_QueryPower

Returns the value calculated by the specified power measurement.

Arguments
IN int pwrsrc -power measurement to take
OUT float *value -location containing voltage/current value on return

Returns
ACS_SUCCESS
ACS_FAILURE

### 13.1.2 ACS_PwrManConfigure

Configures one of the Power Management states.

Arguments
IN int pwrstate -Power state affected by configure
IN int enblmask -Devices to be enabled

Returns
ACS_SUCCESS
ACS_FAILURE

### 13.1.3 ACS_PwrManRead (see note below)

Reads the Power Control Register.

Arguments
OUT int *value -location containing the value of the Power Control Register

Returns
ACS_SUCCESS
ACS_FAILURE

### 13.1.4 ACS_PwrManWrite (see note below)

Writes to the Power Control Register.

Arguments
IN int value -value to be written to the Power Control Register

Returns
ACS_SUCCESS
ACS_FAILURE

**Note:** The ACS_PwrManRead and ACS_PwrManWrite functions could be added to the read and write register commands.

#### 13.1.4.1 Temperature Monitor Commands

### 13.1.5 ACS_QueryTemp

Returns the actual temperature of the specified Xilinx part.

Arguments

| | | |
|---|---|---|
| IN | int partID | -Xilinx part of interest |
| OUT | int *temp | -location containing temperature on return |

Returns

ACS_SUCCESS
ACS_FAILURE

### 13.1.6 ACS_SetTempThresholds

Sets warning and shutdown temperature thresholds. This function enables the temperature monitor operation which is disabled by default.

Arguments

| | | |
|---|---|---|
| IN | int warn | -Temperature warning threshold |
| IN | int shutdown | -Temperature shutdown threshold |

Returns

ACS_SUCCESS
ACS_FAILURE

### 13.1.7 ACS_DisableTemp

Disables the temperature monitor operation.

Returns

ACS_SUCCESS
ACS_FAILURE

### 13.1.8 ACS_ResetTemp

Clears the status of the temperature monitor previously reported. Resets the temperature monitor operation.

Arguments

| | | |
|---|---|---|
| IN | int warn | -Warning status bits to reset |
| IN | int shutdown | -Shutdown status bits to reset |

Returns

ACS_SUCCESS
ACS_FAILURE

### 13.1.9 ACS_QueryTempStatus

Returns the status of the temperature monitor. May be used to determine the Xilinx part that caused a temperature monitor interrupt.

Arguments

| | | |
|---|---|---|
| OUT | int *warnMask | -Location containing the warning status on return |
| OUT | int *shutMask | -Location containing the shutdown status on return |

Returns
ACS_SUCCESS
ACS_FAILURE

### 13.1.10 ACS_QueryTempThresh

Returns the current settings of the temperature monitor thresholds. May be used to determine the Xilinx part that caused a temperature monitor interrupt.

Arguments

| | | |
|---|---|---|
| OUT | int *warnThresh | -Location containing the warning threshold on return |
| OUT | int *shutThresh | -Location containing the shutdown threshold on return |

Returns
ACS_SUCCESS
ACS_FAILURE

**Power Monitor Functions**

Power Monitor functions include:

Query Power

- Returns the value calculated by the specified power measurement on the specified board
- Currently supported on both the Wildstar board and the Firebird board
- On the Wildstar, the following power measurements are available
  - Current being supplied by the system's 3.3V supply
  - Current being used by the on-board 2.5V regulator
  - Current being used by the on-board 2.5/1.8V regulator
  - Current being used by the on-board 3.3V regulator
  - Voltage level on the +5V plane
  - Voltage level on the 3.3V plane
  - Voltage level on the 2.5 V plane
  - Voltage level on the 2.5/1.8V plane
- On the Firebird, the following power measurements are available
  - System 5V current used by PE0 core
  - System 5V current used by PCI core
  - Current used by Firebird +5V plane
  - PE0 core voltage
  - 3.3V current used by PE0
  - Current provided by system on 3.3V pins
  - PCI Controller core voltage
  - System 5V current used by IOPE core
  - 5V current used by IO board
  - IO PE core voltage
  - Voltage on 3.3V plane
  - 3.3V current used by IO board
  - 3.3V auxiliary current used by PLD
  - 3.3V AUX voltage
  - Voltage on 5V plane

Configure Power Management State

- Configures one of the Power Management states on the specified board
- Currently supported only on the Firebird board
- Power Management States available for configuration include
  - Full Power

    - Mid Power
    - Low Power
- Devices available for enable in each state include
    - Clock
    - I/O
    - PE
    - Memory 0
    - Memory 1
    - Memory 2

Read Power Control Register

- Reads the Power Control Register on the specified board
- Currently supported only on the Firebird board
- **This function could be added to the Read Register function already existing in the SLAAC API**

Write Power Control Register

- Writes desired value to the Power Control Register on the specified board
- Currently supported only on the Firebird board
- **This function could be added to the Write Register function already existing in the SLAAC API**

**Temperature Monitor Functions**

The temperature of various Xilinx components on the Wildstar boards may be monitored. The user specifies temperature thresholds for both warning and shutdown levels. Once the warning threshold for a part has been exceeded, the status bit for the part is set and an interrupt is generated. If the shutdown threshold for a part has been exceeded, the part is immediately deprogrammed by the hardware.

Temperature Monitor functions include:

Query Temperature

- Returns the actual temperature of the specified Xilinx part
- Currently supported on both the Wildstar board and the Firebird board
- On the Wildstar, the temperature of the following parts may be queried
    - PCI Controller
    - Processing Element 0
    - Processing Element 1
    - Processing Element 2
    - External I/O 0
    - External I/O 1
    - Mezzanine Connection 0
    - Mezzanine Connection 1
- On the Firebird, the temperature of the following parts may be queried
    - PCI Controller
    - Processing Element 0
    - External I/O 0
    - Relative ambient temperature near PE0
    - Relative ambient temperature near board power regulators

Set Temperature Thresholds

- Sets warning and shutdown temperature thresholds on the specified board
- The setting of the temperature thresholds enables the temperature monitoring function

Disable Temperature Monitoring

- Disables the temperature monitoring function on the specified board

Reset Temperature Monitor

- Clears the status of the temperature monitor previously reported and resets the temperature monitor function on the specified components

Query Temperature Monitor Status

- Returns the warning and shutdown status on the specified board
- This function may be used to determine which Xilinx part caused a temperature monitor interrupt

Query Temperature Monitor Thresholds

- Returns the current warning and shutdown temperature settings on the specified board
- This function may be used to determine which Xilinx part caused a temperature monitor interrupt

DMA Functions

On the Wildstar and Firebird boards, DMA transfers are supported. These functions include:

DMA Memory Buffer Allocation

- Allocates a buffer to be used for DMA operations
- The buffer is usually contiguous
- Not supported under Solaris OS

DMA Memory Buffer Free

- Frees a DMA buffer previously allocated
- Not supported under Solaris OS

Set DMA Resource Timeout

- Sets the amount of time to wait for a PE resource to become available when initiating a DMA transfer

DMA Read

- Reads a buffer of data from the specified PE using DMA hardware

DMA Write

- Reads a buffer of data from the specified PE using DMA hardware

**Interrupt Functions**

There are 3 Interrupt functions defined in the SLAAC API (ACS_Interrupt, ACS_InterruptPoll, and ACS_InterruptAck). Currently, none of them are implemented for the Wildstar boards.

The Wildstar and Firebird boards may generate interrupts from the Processing Elements (including those on external I/O boards), the Temperature Monitor, and the DMA Controller.

The Interrupt functions supported by the Wildstar API include:

Reset Interrupt

- Resets the specified interrupt sources on the board
- The resetting of the interrupt enables the specified interrupt to begin generating interrupts

Wait On Interrupt

- Waits for an interrupt from any of the specified interrupt sources on the specified board

Query Interrupt Status

- Returns the status of pending interrupts on the specified board
- Interrupt sources include PE interrupts as well as temperature monitor interrupts

The SLAAC 2 API contains functions for System Setup, Memory Access, Register Access, Interrupt, Streaming Data, Convenience, System Management, Group Management, and Non-Blocking.

The Memory Access, Register Access, Interrupt, Convenience, and Streaming Data functions are customized on a platform basis. Provisions are made for functions that are not supported by a given platform.

The SLAAC 2 API currently supports the following platforms:

- Wild Force 4
- Wild Force 4F  (WildForce with VT Fast Fifos)
- Wildstar
- RCM
- SLAAC 1
- SLAAC 1V

DMA, Power Monitoring, and Temperature Monitoring functions are candidates for addition to the SLAAC 2 API.  In order to support the DMA and Temperature Monitoring functions, the Interrupt functions on the Wildstar platform would need to be implemented as well.

## 14 Appendix 3 – Sample Application Registry

```
APPNAME  Radio

# Application Registry for Processor in FPGA
# Times and Power are for FPGA related processing only
# 100 Mhz clock frequency
#
#  NAME  IDNum Quality  Watts
#  ----- ----- ----- -----
#MODE PN_16 1  1  0.0
#MODE PN_32 2  2  0.0
MODE  PN_64 3  3  14.6
MODE  PN_128   4  4  15.7
MODE  PN_256   5  5  16.8
MODE  PN_512   6  6  16.8
MODE  PN_1024  7  7  17.1


#
#  FROM  TO Watts Seconds
#  ----- ----- ----- -----
RETOOL   *  PN_64 13.8   .7
RETOOL   *  PN_128   13.8  .8
RETOOL   *  PN_256   13.8  1.1
RETOOL   *  PN_512   14.1  5.1
RETOOL   *  PN_1024  14.4  9.6


MEASURE  Q_Delta     Requested Quality Delta
MEASURE  Fatal_Error At least one message had a fatal error
MEASURE  Desired_BER Desired Bit Error Rate
MEASURE  Actual_BER  Actual Bit Error Rate
```

# 15 Appendix 4 – Messages for tcp/ip sockets

## 15.1 Receiver (application) to Application Manager Messages

Int “Q_Delta” values 0..1
Int ”Fatal_Error ” values 0..1
Float “Actual_BER” %0.6f
Or
String “Actual_BER” “N/A”
Int “Desired_BER” values –3..-9

## 15.2 Statistics to Simulation GUI Messages

For every message received:
Int “Mode” 16, 32, 64, 128, 256, 512, 1024
Int ”Smode” -3, -4, -5, -6, -7, -8, -9
Int “MBE” values 0..? (# bit errors in current message)
String “Code” “NAL” (No Acquisition Lock found)
“NFS” (No Frame Sync found)
“ACQL” (Acquisition Lost during message processing)
“NEOM” (No EOM found)
“IM” (Incomplete Message)
“CM” (Corrupt Message)
“GM” (Good Message)
String “MSG” message itself

Note: Not all possible messages will be sent to the GUI each message receipt.

At each statistics interval:
Int “TBITS” values 0..? (Total number of bits transmitted in interval)
Int “BERR”values 0..? (Number of bit errors in interval)
Float ”ABER”%0.6f
Or
String “ABER”“N/A”

Every block of messages will end with the EOB message:
String “EOB” “EOB”

# 16 Appendix 5 – Software Radio Mode/Submode Information

## 16.1 Application Version (Modes)

PN16 (PN code size 16) – lowest power consumption
PN32 (PN code size 32)
PN64 (PN code size 64)
PN128 (PN code size 128)
PN256 (PN code size 256)
PN512 (PN code size 256)
PN1024 (PN code size 1024) – highest power consumption

## 16.2 Submodes

BET9 (Bit Error Threshold $10^{-9}$) – lowest probability of overcoming fatal error vs. lowest bit error rate
BET8 (Bit Error Threshold $10^{-8}$)
BET7 (Bit Error Threshold $10^{-7}$) - default
BET6 (Bit Error Threshold $10^{-6}$)
BET5 (Bit Error Threshold $10^{-5}$)
BET4 (Bit Error Threshold $10^{-4}$)
BET3 (Bit Error Threshold $10^{-3}$) – highest probability of overcoming fatal error vs. highest bit error rate

Each mode contains each submode

Mode change

- Causes retooling (recalculating reference codes, re-stabilizing the receiver, reloading FPGA core, etc)
- Costs power to retool
- Impacts continuous power usage (higher PN code size uses more power)

Submode change does not impact power usage

Fatal errors

- No acquisition lock
- No frame sync found
- Acquisition lock lost during message processing
- No End of Message (EOM) found
- Incomplete message

Non-fatal conditions

- Unacceptable bit error rate

Modes changes are a “gross” adjustment.
Submode changes are a “fine” adjustment.

If a fatal error occurs, lowering the submode (bit error rate threshold) increases the probability of receiving a full message, but also increases the bit errors in the message. In other words, a submode of BET3 within any mode is the best chance at receiving a complete message, but also will have a higher number of bit errors if a full message is received whereas a submode of BET9 decreases the chance of receiving a full message, but will have a lower number of bit errors if a full message is received.

The Application will make a mode change recommendation to the Application Manager. The Application Manager will assess the recommendation using the Application Registry and the Mission Profile and respond with a mode command.

The Application will manage submode changes as there is no need for the Application Manager to approve submode changes.

### 16.3 Mode/Submode Rules for the Application

```
If a fatal error occurs and the submode is not BET3
    Change submode to next lowest submode
    Recommendation to AM: Leave mode as is

If a fatal error occurs and the submode is BET3
    Recommendation to AM: Change mode to next highest mode
    Default submode to BET7 within new mode

If the bit error rate is unacceptable and the submode is not BET9
    If fatal error at next highest submode has not occurred in current mode more than once
        Change submode to next highest submode
        Recommendation to AM: Leave mode as is
    Else
        Recommendation to AM: Change mode to next highest mode
        Default submode to BET7 within new mode

If the bit error rate is unacceptable and the submode is BET9
    Recommendation to AM: Change mode to next highest mode
    Default submode to BET7 within new mode
```